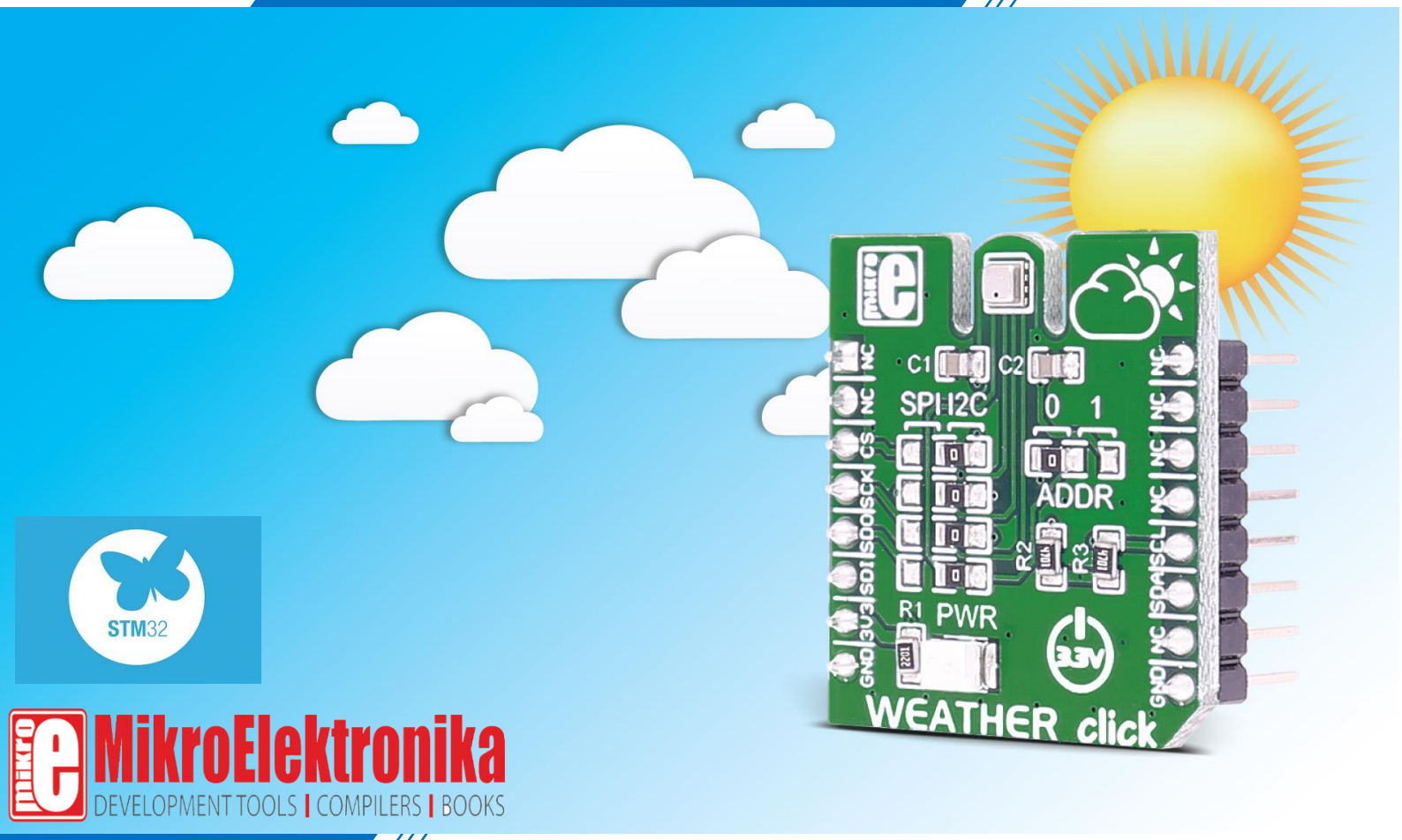
BORDEAUX YNOV CAMPUS

2020

Projet Weather Click



Jean-Gabriel Massicot
Axel Devaux

GROUP REPORT

Sommaire

Introduction

Le projet *WeatherClick* est un projet qui consiste à utiliser un capteur relevant la température, l'humidité et la pression atmosphérique, à récupérer les valeurs, à les retransmettre via une liaison série afin de les afficher pour l'utilisateur.

Cela est mis en œuvre à l'aide d'un STM32F407, d'un *shield* de développement avec des slots au standard *mikroBUS*, et d'un module *WeatherClick mikroBUS* comportant les capteurs. Le *WeatherCLick* communique avec la STM32 par I2C. La carte STM32 est reliée par liaison série USART à un ordinateur.

Ce rapport résume d'une part la partie affichage graphique réalisée en Java et d'autre part la partie driver du module *WeatherClick* réalisée pour la STM32 en C.

Partie Graphique

Emplacements

Les fichiers .java sont disponibles dans les repository *GitHub* suivant :

- https://github.com/massicotjgab/Station_meteo_WeatherClick
- https://github.com/DvAx/Station_meteo

Plus précisément dans :

- Src/GRAPHIC/WeatherClick/src

Le fichier texte est à l'emplacement :

- Src/GRAPHIC/WeatherClick/bin/Values.txt

Un fichier README qui donne les informations sur l'environnement et l'usage est à l'emplacement :

- Src/GRAPHIC/README.md

Choix du langage

Pour réaliser la partie graphique nous avons eu du mal à choisir une technologie. Nous ne maitrisions pas forcément ce domaine avant ce projet.

Dans un premier temps un affichage console basique en script shell était prévu. Cependant après quelques essais, il en est ressorti que cette solution était inesthétique, pas pratique d'utilisation, et demandait un peu trop de recherches. Nous avons donc décidé de laisser de côté la partie graphique pour se concentrer sur la partie Driver.

Après la fin du projet nous avions l'idée de réaliser l'interface en qml et en Python. D'autant plus qu'un autre projet réalisé depuis dans ces langages nous avait permis de monter en compétences sur le sujet.

Entre temps, la limite de temps que nous pensions proche s'est avérée être reportée au 8 Avril ce qui nous a permis de différer encore un peu la réalisation. Grand bien nous en a pris puisque nous avons entre temps démarré des cours de Java, basés essentiellement sur de la réalisation d'Applets et de Frames, c'est-à-dire de représentations graphiques.

Nous étions tous les deux dans ce cours, et plus le cours avançait plus nous étudiions précisément tous les éléments dont nous avions besoin. Le choix du langage Java pour réaliser l'interface graphique s'est alors imposé à nous.

Architecture & Fonctionnement

Le projet est constitué de deux classes Java principale et d'une classe permettant la fermeture de Frames. Le fichier *Affiche_Data.java* est celui où est instancié la Frame, tandis que le fichier *Forme.java* est celui de l'applet. Les classes correspondantes portent le même nom que les fichiers.

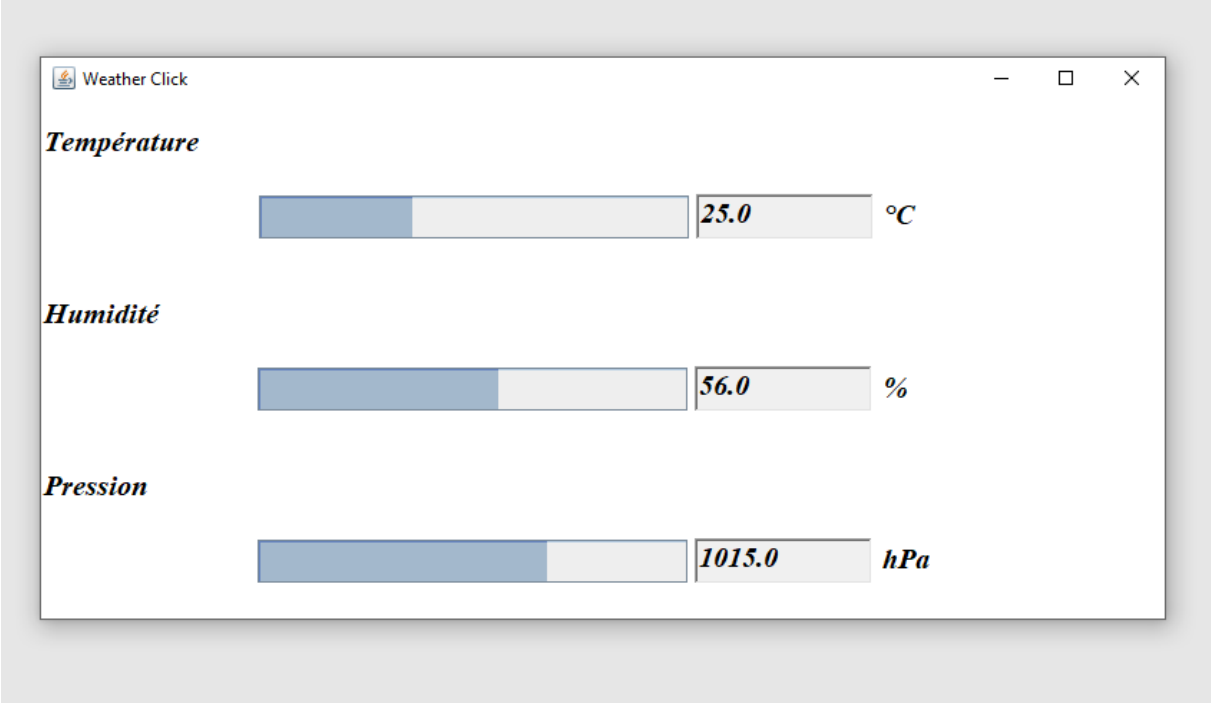
La Frame sert essentiellement à exécuter la partie Applet dans laquelle les éléments sont organisés, et à mettre le tout dans une fenêtre. L'applet gère la création et le positionnement des objets, les méthodes permettant la modification des objets (ProgressBar, TextField), la méthode de lecture des valeurs et le thread qui permet de mettre à jour le graphique en temps réel.

Les températures sont prévues en degré Celsius, l'humidité en pourcentages et la pression en hectopascals.

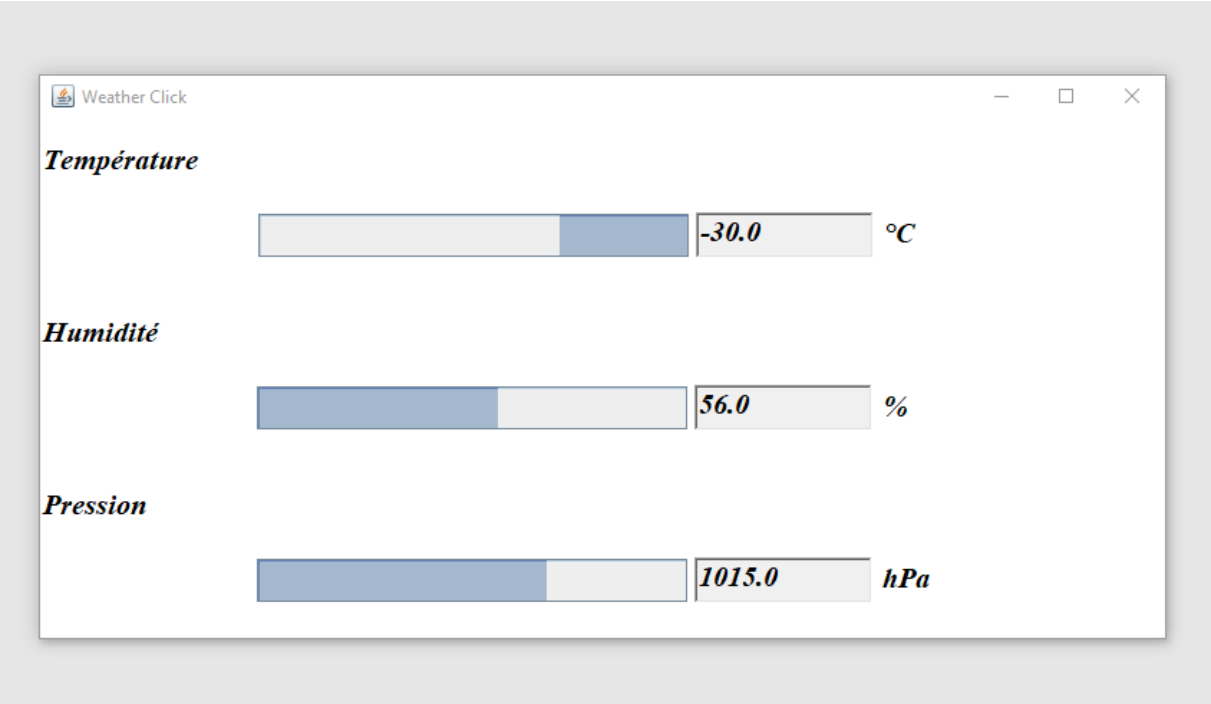
Les plages d'évolution ont été choisies en fonction des valeurs records mondiales légèrement majorées (arrondies au supérieur). L'humidité est comprise entre 0 et 100%, bien entendu. Ces plages sont les suivantes :

- Température négatives : -100 <- 0 °C
- Températures positives : 0 -> 70 °C
- Humidité : 0 -> 100 %
- Pression : 860 -> 1090 hPa

Ci-dessous on peut voir la façon dont cela est représenté graphiquement :



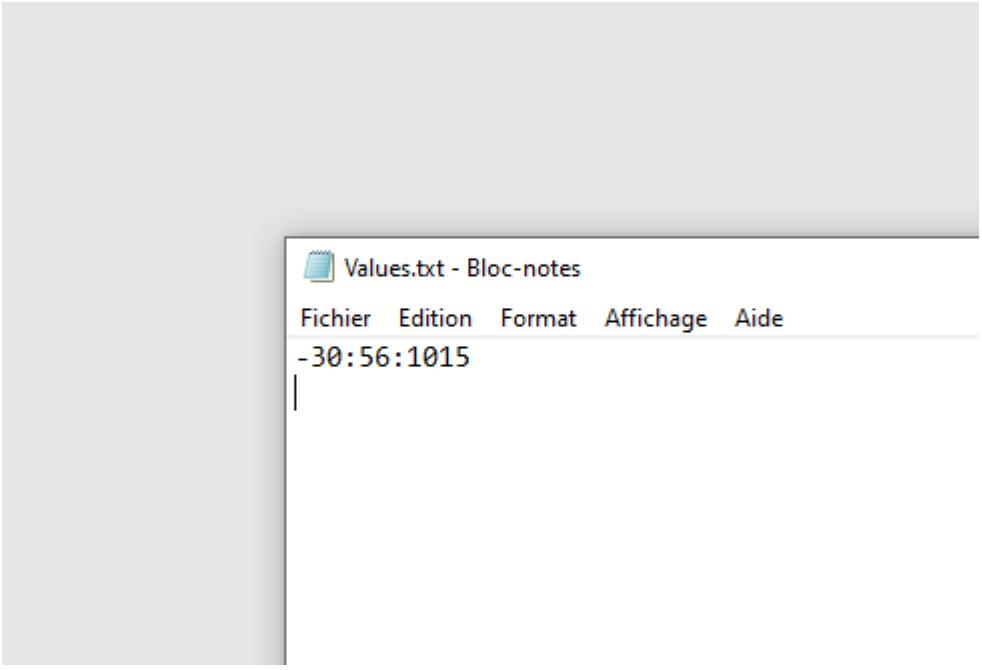
Lorsque la température est négative l'interface graphique se modifie automatiquement pour que la barre de température soit orientée de droite à gauche, comme sur la représentation ci-dessous :



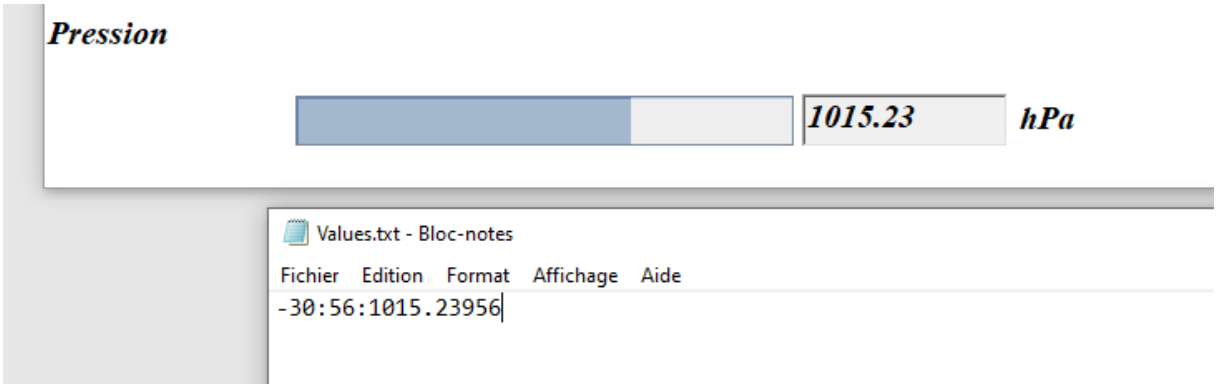
Actuellement l'application Java va lire dans un fichier texte les valeurs. Celles-ci sont de la forme suivante :

Température (en °C):Humidité (en %):Pression (en hPa)

Le fichier texte ne comporte qu'une ligne. Il simule la trame série envoyée par la partie série par l'USART. L'exemple ci-dessus correspond donc au contenu de fichier ci-dessous :



Ces valeurs sont des doubles donc peuvent être à virgule (un « . » (dot) représente la virgule). Ces valeurs sont cependant castées en integer pour modifier les ProgressBar et tronquées au deuxième chiffre après la virgule pour l'affichage, comme dans l'exemple ci-dessous :



Le thread lit le fichier toutes les 100 ms et rafraîchit alors l'interface graphique. Donc généralement lors d'une modification, au moment où l'on enregistre le fichier (à l'aide d'un (ctrl + s) par exemple), l'interface se modifie instantanément.

En théorie une lecture série est effectuée et vient modifier soit le fichier (solution préférée) soit l'interface directement dans le thread, à la place de la lecture du fichier. Les données sont donc sensées être envoyées au même format.

Problèmes rencontrés

Lors du développement, nous n'avons pas réussi à utiliser le port série en Java. Cela demandait sans doute un peu plus de connaissances que nous n'en avions. Nous avons pensé faire un script Shell ou Python pour faire le lien, lire le port Série et modifier le fichier du programme Java. Cependant nous n'avons pas encore pu le mettre en place.

Le projet a été en quelque sorte évolutif, car nous continuions à apprendre des éléments, et au début la modification de la partie graphique ne marchait pas. L'apprentissage des Threads lié à celui de la lecture de fichier et du découpage de chaines de caractères nous a permis de corriger les soucis que nous avions à ce niveau-là.

Notions Apprises

La partie graphique nous a permis de nous perfectionner en langage orienté objet, et nous a permis de mettre en œuvre beaucoup d'éléments que nous avions appris en cours de Java et regroupe la majorité des concepts étudiés. Cela a été donc plutôt intéressant pour nous. Les notions de représentation graphique, de lecture de fichier, et de Thread nous ont particulièrement intéressé et servi.

Programmation sur STM32F407

Emplacements

Les codes sources sont disponibles dans les repository *GitHub* suivant :

- https://github.com/massicotjgab/Station_meteo_WeatherClick
- https://github.com/DvAx/Station_meteo

Le fichier .C source que nous avons utilisé est plus précisément à l'emplacement suivant :

- /Src/STM32/Src/main.c

Présentation

Cette partie du projet porte sur la programmation de la STM32F407 afin de nous permettre de récupérer les mesures faites par le capteur présent sur le *WeatherClick*.

Ce capteur est un capteur BME280 qui peut mesurer trois grandeurs physiques qui sont la température, le taux d'humidité et la pression.

Le capteur se présente sous la forme suivante :

Cette petite carte au standard *mikroBUS* se présente sous la forme d'une petite carte de dimensions 19.0mm x 18.0mm x 3.0mm d'environs 1.0g. Il s'agit donc d'une solution compacte compatible avec l'environnement des systèmes embarqués. Elle est alimentée en 3.3V ou 5V.

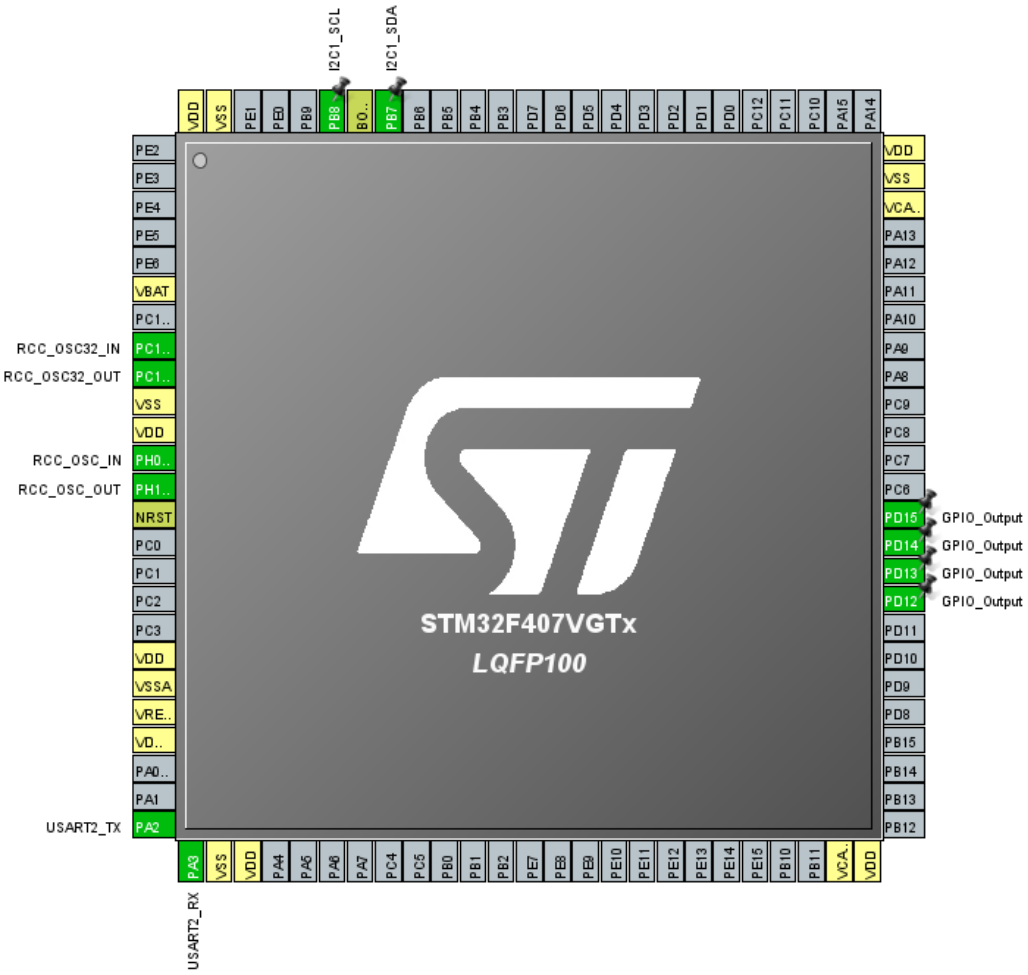
Choix du BUS de communication

En ce qui concerne le projet que nous devions réaliser, la caractéristique qui nous intéresse et qui a impacté la façon de programmer la STM32 est le mode de communication. Il possède en effet un bus I2C et un bus SPI, et l'un des ports doit être choisi.

Nous avons donc dû choisir l'un des deux bus pour communiquer, et notre choix s'est porté sur l'I2C. En effet le mode est sélectionné au moyen d'un cavalier. Sur la carte *WeatherClick* les cavaliers ont été remplacés par de petites résistances soudées. Comme on peut le voir sur la photo ci-dessus, le bus de communication présélectionné est l'I2C, et nous ne souhaitions pas prendre le risque de dessouder les quatre composants pour les ressouder du côté SPI.

Partie CubeMX

Pour paramétrer le code, nous avons décidé d'utiliser le logiciel CubeMX qui permet de prévoir graphiquement les broches utilisées et offre un environnement de PAO (Programmation Assisté par Ordinateur) qui couvre beaucoup de possibilités. Grâce à ce logiciel nous avons configuré le microcontrôleur de la façon suivante :



On peut voir que nous avons configuré le microcontrôleur pour pouvoir communiquer en I2C et en USART (Pin A2 et A3 pour l'USART et Pin 87 et 88 pour l'I2C). Ainsi que 4 GPIO en Output. Les GPIO avaient été prévues pour les LEDS : PD15 (LED bleue), PD14 (LED rouge), PD13 (LED orange) et PD12 (LED verte). Cela devait servir à visualiser des passages de données, mais nous ne les avons finalement pas utilisées.

L'USART est plutôt facile à implémenter sur la STM32 grâce à CubeMX et il s'agit d'un moyen pratique de communiquer avec un ordinateur doté d'un logiciel tel que Putty, TeraTerm, ou Hercules, ou même au moyen de commandes Shell ou Python. Cela nous permettait donc de développer et de faire des essais même sans partie graphique. Par ailleurs nous nous sommes dit que ce serait surement le moyen de communication le plus utilisable par le code Java. C'est donc le mode que nous avons choisi pour transmettre les données depuis la STM32 vers l'ordinateur.

Code et programmation

Le code disponible sur *GitHub* a été en partie auto-généré par le logiciel CubeMX, comme expliqué ci-dessus, et en partie programmé par nos soins sur l'IDE Keil.

Pour la partie I2C, la STM32 était considéré comme Master et le module *WeatherClick* comme Esclave. Les fonctions utilisées sont des fonctions HAL. Les plus importantes sont **HAL_i2C_Master_Transmit** et **HAL_i2C_Master_Receive.** Pour la transmission USART nous avons également utilisé des fonctions HAL, surtout la fonction **HAL_UART_Transmit.**

Nous avons réussi à communiquer avec le capteur. Nous voulions tout d'abord effectuer un teste qui était de lire l'ID situé au registre 0xD0 de la memory map. La memory map et représentée ci-dessous :

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1…0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88…0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

En effet lorsque nous allons lire dans ce registre nous obtenons bien la réponse attendue qui est 0x60. Cependant lorsque nous avons procédé de la même manière pour lire la température, la pression et l'humidité nous n'avons pas réussi à avoir de résultat. A titre d'exemple, pour la température nous recevions « 226 130 172 » en décimal.

Nous pensons que ce dysfonctionnement est dû à une mauvaise gestion de la librairie qui permet normalement de faire les compensations nécessaires afin de pouvoir lire correctement les trois valeurs qui nous intéressent.

Si nous avions pu récupérer les bonnes informations, nous aurions pu les traiter de deux façons différentes. La première méthode aurait consisté à stocker les valeurs de chaque grandeur physique indépendamment dans trois tableaux différents, à les convertir et à formater une chaîne de caractère dans le format attendu par la partie graphique. Ensuite nous aurions envoyé cette chaîne de caractères pas la liaison série.

La deuxième méthode aurait été d'envoyer directement les données reçues par la STM32 et de modifier le code Java de la partie graphique pour faire la conversion et récupérer les valeurs directement depuis l'ordinateur, ce qui aurai sans doute facilité le calcul, mais aurai été moins pertinent dans un projet ou le but était de développer un driver sur la STM32 pour le capteur.

Nous avions donc privilégié la première proposition, puisque de cette façon, même sans l'interface graphique, les données auraient été humainement interprétables en utilisant simplement un logiciel de liaison série comme Putty. Par ailleurs au moment du développement nous n'avions pas encore choisi le langage qui serait utilisé pour la partie Graphique. Ce choix a donc aussi été fait pour être compatible et facilement interprétable quel que soit le langage utilisé pour l'interface.

Zephyr Project

Nous avons aussi essayé de faire le projet d'une deuxième façon : en utilisant Le Projet Zephyr. Ce projet de la *Linux Foundation* est un projet Open Source et donc collaboratif. Il permet de mettre en place un OS temp réel RTOS optimisé pour un certain nombre de plateforme. La méthode peut faire penser au Projet Yocto sur certains aspects bien que ce soit très différent car orienté microcontrôleur.

Ce projet prend en charge de nombreuses cartes et de nombreux modules et capteurs. Parmi ces éléments supportés, la carte STM32F407 utilisée pour ce projet et le capteur BME280 étaient disponibles.

Nous avons réussi à mettre en place l'environnement de compilation et à faire de premiers essais. Malheureusement cela n'a pas fonctionné et comme il s'agissait d'un projet en C nous avons décidé de ne pas passer trop de temps sur cette méthode.

Problèmes rencontrés

Le premier problème que nous avons rencontré, a été qu'aucun de nous deux n'est vraiment spécialiste des drivers et des langages bas-niveaux. Même si nous maitrisons le langage C, l'utiliser au sein de codes généré par des logiciels orientés ARM comme CubeMX, et envisager les multiples fonctions à disposition comme les fonctions HAL n'a pas été simple.

Cela est dû également à un cruel manque de pratique, de cours ou de projets en langage C. C'était la première fois de l'année de Mastère 2 où nous avions à faire un projet orienté système embarqué en C.

Au sujet du Projet Zephyr, nous connaissions assez peu le système et nous avons eu des difficultés pour mettre en place l'environnement. Finalement nous n'avons pas réussi à l'utiliser. Cependant cela nous a ouvert beaucoup de perspectives, et l'aspect Open Source et RTOS sous environnement Haut-Niveau nous a beaucoup intéressé.

Notions Apprises

Dans ce projet nous avons pu découvrir le module WeatherClick. Nous avons enrichi nos connaissances de CubeMx et des fonctions HAL. Ce projet nous a aussi permis de pratiquer un peu de langage C ce qui nous a été appréciable.

Nous avons aussi découvert le Projet Zephyr de la *Linux Foundation* qui, bien que peu adapté pour ce projet, nous a beaucoup intéressé pour la suite.

Conclusion du projet

Ce projet nous a beaucoup intéressé, avec une préférence pour la partie graphique en Java. Malgré tout pratiquer un peu en langage C nous a bien plu. Malgré quelques difficultés, nous avons pu approfondir diverses notions, depuis celles d'orienté-objet jusqu'aux fonctions HAL.

Bien que le format du projet (très court, en autonomie sans professeur, et sur un sujet qui semblait avoir déjà était réalisé dans d'autres projets sur Internet) ne correspondît pas pour nous à une situation efficace pour apprendre, nous avons tout de même réussi à mettre à profit ce projet.

La découverte de Zephyr Project a été la bonne surprise de ce projet, même si de prime-abord, la prise en main nous paraît assez complexe.

C'est cette amplitude d'éléments abordés qui a fait de ce projet un projet complet. Cela nous a permis de mettre en œuvre des éléments appris dans différents cours, ce qui est plutôt une bonne chose pour un projet d'étude.