

École Publique d'Ingénieurs en 3 ans

Rapport de projet industriel

# PROJET « I AM ENSICAEN »

le 7 février 2022

Lucile VOISIN  
[lucile.voisin@ecole.ensicaen.fr](mailto:lucile.voisin@ecole.ensicaen.fr)

Tuteurs école :  
Hugo DESCORBES  
Philippe LEFEBVRE



[www.ensicaen.fr](http://www.ensicaen.fr)

# TABLE DES MATIÈRES

---

<b>INTRODUCTION</b>	<b>3</b>
<b>1. PRÉSENTATION DU PROJET</b>	<b>3</b>
<b>2. PRÉSENTATION DU SOUS-SYSTÈME</b>	<b>4</b>
2.1. Place et rôle du sous-système dans le projet « I am Ensicaen »	4
2.2. Architecture matérielle du sous-système	5
2.3. Matériel	6
2.3.1. Objet connecté	6
2.3.2. Gateway	7
2.3.3. Branchements	8
<b>THINGS</b>	<b>9</b>
<b>1. APPLICATIONS DE L'OBJET CONNECTÉ</b>	<b>9</b>
<b>2. MODULES LORA ÉTUDIÉS</b>	<b>10</b>
2.1. Module RL01	11
2.2. Module CR02	17
2.2.1. Mesure et mise à disposition des données pour l'émetteur	17
2.2.2. Émission des données par communication LoRa	18
<b>GATEWAY</b>	<b>19</b>
<b>1. APPLICATIONS DE LA GATEWAY</b>	<b>19</b>
<b>2. RÉCEPTION DES DONNÉES</b>	<b>20</b>
2.1. Module CR02 récepteur	20
2.2. Lecture du port série	21
<b>3. TRANSMISSION DES DONNÉES VERS UN BROKER</b>	<b>23</b>
3.1. Protocole MQTT	23
3.2. ThingSpeak	23
3.3. Transmission des données en deux temps	24
3.3.1. Première étape : serial_to_localhost.c	24
3.3.2. Deuxième étape : localhost_to_ThingSpeak.py	25
<b>CONCLUSION</b>	<b>27</b>
<b>ANNEXES</b>	<b>28</b>

# INTRODUCTION

---

*Le projet industriel se veut dans la continuité du projet immersif « I am Ensicaen » dont la vision à terme est de développer des solutions de mesure nomades sur batterie et connectées par protocole de communication wireless. L'objectif étant de mesurer le comportement des salles d'enseignement en temps réel pour aider l'utilisateur à comprendre le comportement de son environnement direct de travail afin de lui permettre d'interagir avec lui et d'en avoir un usage plus responsable.*

## 1. Présentation du projet

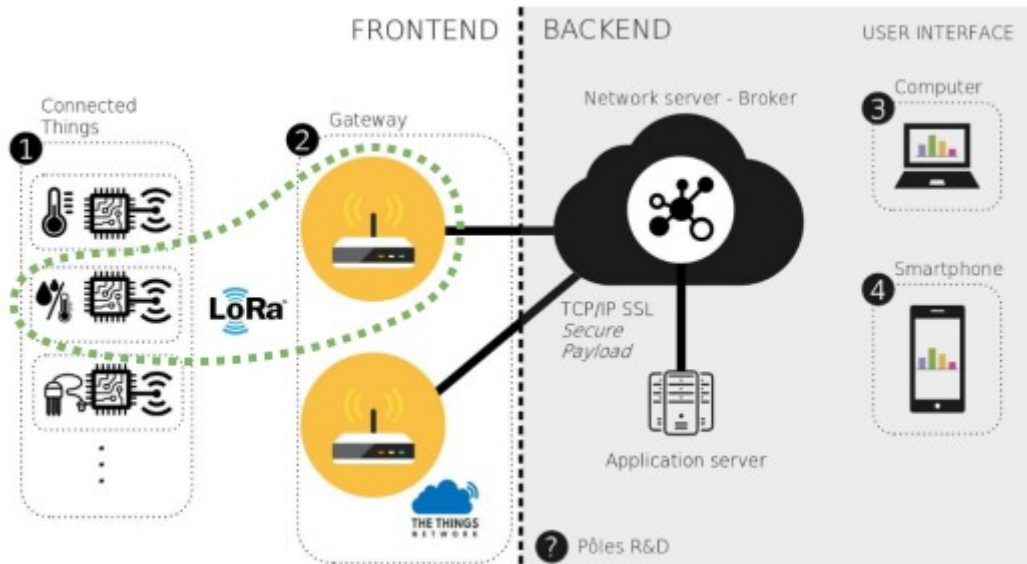
Dans le cadre du projet industriel, le cahier des charges s'est recentré autour de l'objectif de la transmission d'une donnée.

Les développements ont conduit à la réalisation d'un démonstrateur effectuant la transmission des données au travers d'une communication LoRa sans implémentation du protocole LoRaWAN.

La question de l'autonomie énergétique n'a pas été abordée lors de ce projet.

## 2. Présentation du sous-système

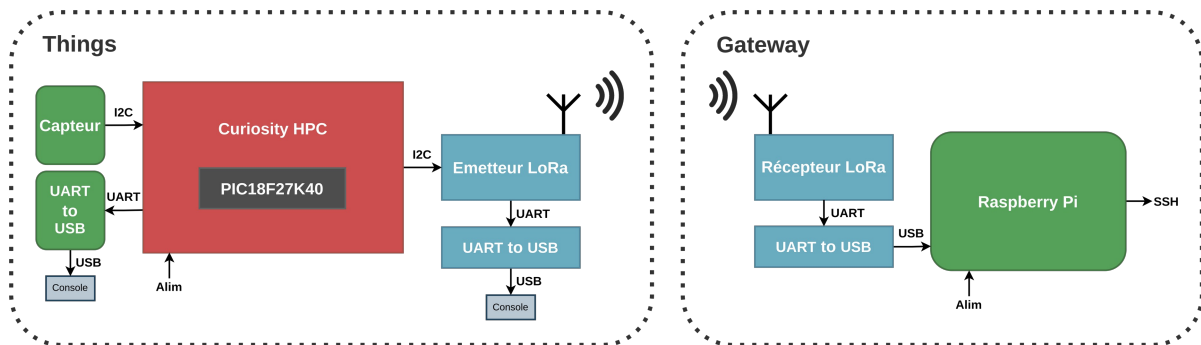
### 2.1. Place et rôle du sous-système dans le projet « I am Ensicaen »



*Illustration 1: Entouré en vert, le sous-système ayant fait l'objet de développements lors du projet industriel*

Le sous-système se compose d'un objet connecté et d'une gateway, ces deux entités sont mises en relation par communication wireless LoRa. Le sous-système constitue la partie frontend du projet « I am Ensicaen » et englobe les fonctionnalités de mesure des grandeurs physiques et de transmission des données dans l'objectif de les mettre à disposition d'un broker.

## 2.2. Architecture matérielle du sous-système



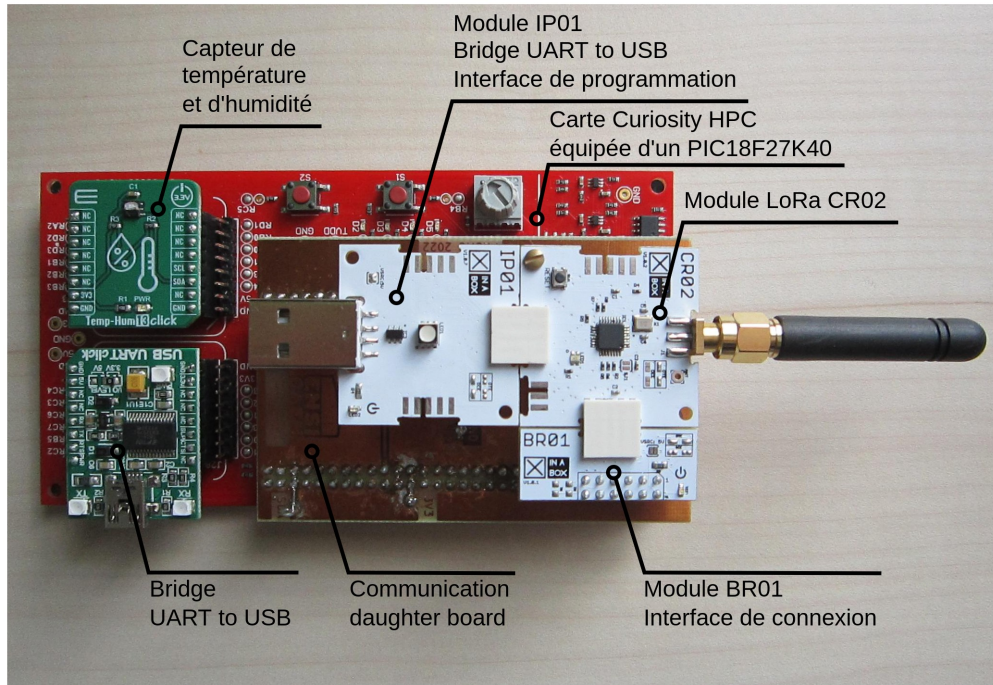
*Illustration 2: Architecture matérielle du sous-système*

L'objet connecté se compose d'un microcontrôleur, d'un capteur et d'un émetteur LoRa. Ces différents composants communiquent entre eux par protocole I2C. Des bridges UART-USB sont également présents pour de l'affichage en console.

La gateway est constituée d'un récepteur LoRa et d'une Raspberry Pi communiquant ensemble par liaison série au moyen d'un bridge UART-USB.

## 2.3. Matériel

### 2.3.1. Objet connecté

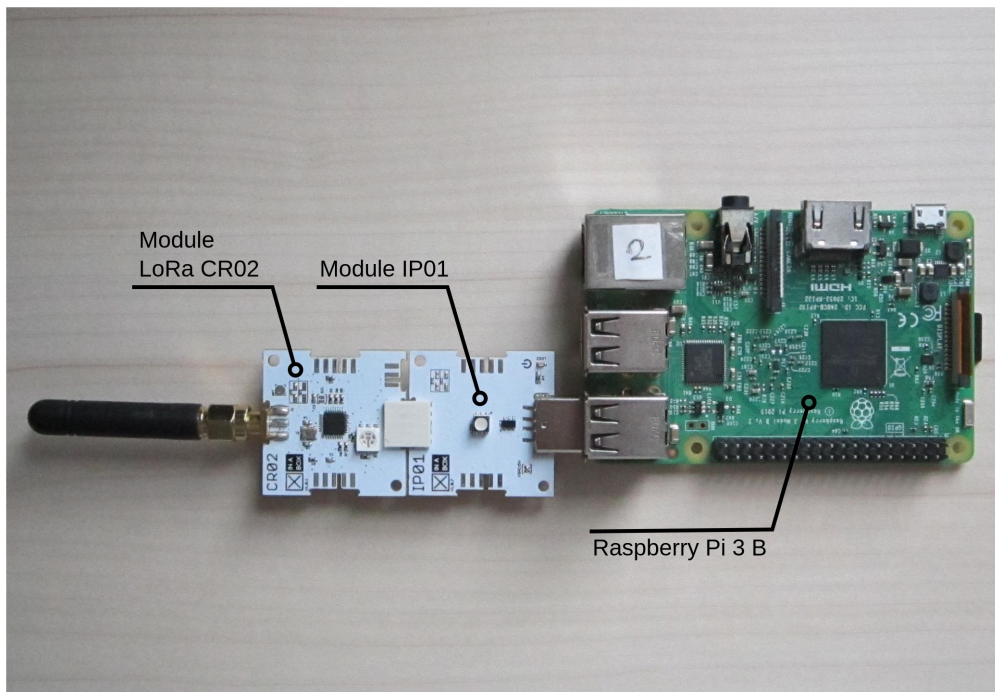


*Illustration 3: Objet connecté*

- Microcontrôleur 8bits PIC18F27K40 Microchip
- Carte de développement Curiosity HPC Microchip
- Clickboard capteur de température et d'humidité Mikroe
- Clickboard bridge UART to USB Mikroe
- Module CR02 LoRa 868 MHz Xinabox utilisé en émetteur
- Module IP01 Xinabox
- Module BR01 Xinabox
- Communication daughter board Ensicaen

Le PIC18 inséré dans la carte Curiosity HPC se situe sous la communication daughter board. La combinaison du module BR01 et de la communication daughter board (composants non représentés sur l'illustration 2) permet de connecter simplement la carte Curiosity avec le module LoRa CR02.

### 2.3.2. Gateway

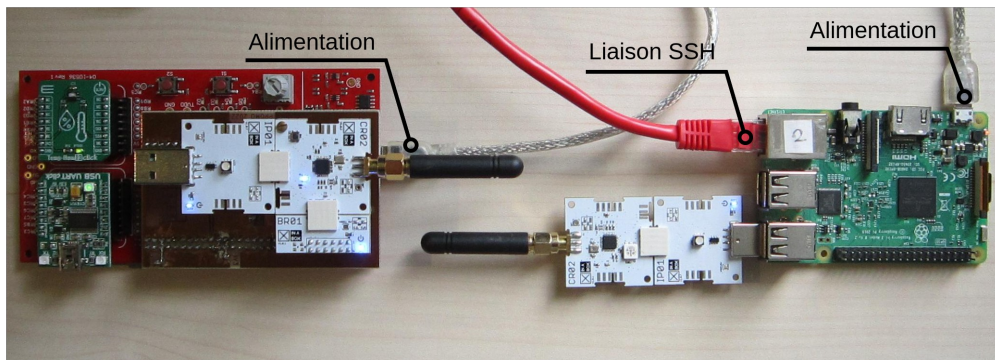


*Illustration 4: Gateway*

- Module CR02 LoRa 868 MHz Xinabox utilisé en récepteur
- Module IP01 Xinabox
- Raspberry Pi 3 B

Le module IP01 sert à connecter le module CR02 sur un port USB de la Raspberry Pi permettant ainsi une liaison série entre ces deux composants. Il est utile également en tant qu'interface de programmation lors des phases de développement.

### 2.3.3. Branchements



*Illustration 5: Branchements*

L'alimentation des cartes est réalisée par câble microUSB-USB. La liaison Ethernet (câble RJ45) permet de connecter la Raspberry Pi à internet et d'interagir avec celle-ci par liaison SSH. Il est également possible de connecter un écran, un clavier et une souris à la Raspberry Pi.

Les modules UART to USB et IP01 de l'objet connecté peuvent être reliés aux ports USB d'un ordinateur, ils sont utiles seulement lors des phases de développement et de debug pour programmer les microcontrôleurs et réaliser des affichages en console via minicom.

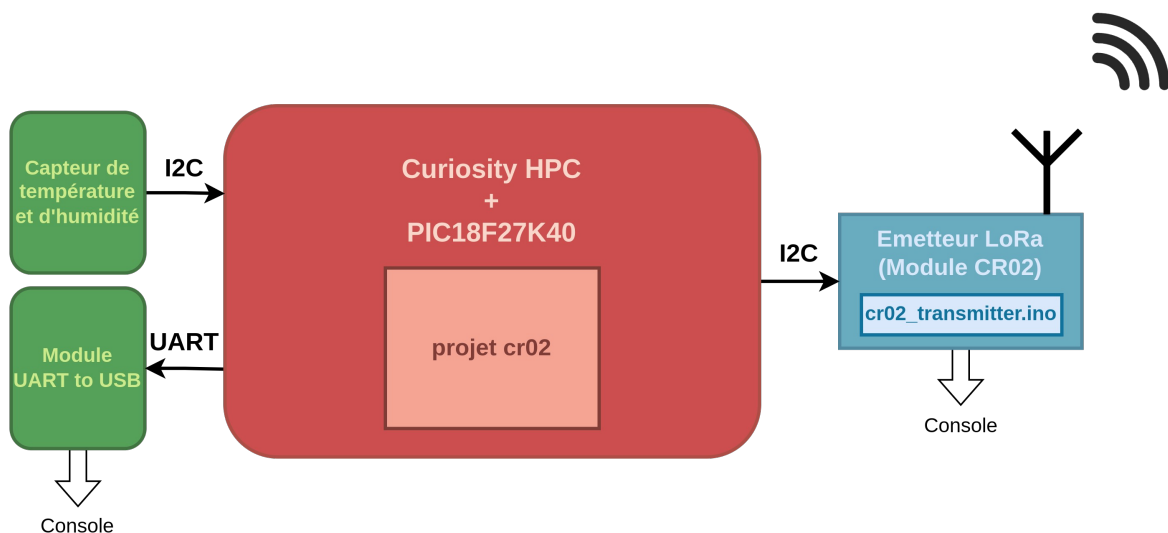


# THINGS

---

*L'objectif de la partie « Things » est l'envoi des données mesurées par un capteur au travers d'une transmission wireless LoRa vers une gateway.*

## 1. Applications de l'objet connecté



*Illustration 6: Applications de l'objet connecté*

L'objet connecté possède deux fonctions, effectuer des mesures et émettre les données par communication LoRa.

La donnée est produite par un capteur de température et d'humidité et mise à disposition de l'émetteur LoRa par un firmware embarqué dans le PIC18, le projet cr02.

La donnée est ensuite émise par l'émetteur LoRa possédant son propre programme, cr02\_transmitter.ino.

## 2. Modules LoRa étudiés

Les deux types de modules ayant été étudiés sont le module RL01 constitué d'un bridge I2C-SPI et d'un transmetteur LoRa 433 MHz et le module CR02 composé d'un microcontrôleur 8bits ATmega328P ainsi que d'un transmetteur LoRa 868 MHz.

Les développements ont été réalisés en langage C sous IDE MPLABX v5.15 et toolchain C XC8 v1.45 de Microchip, pour les projets rl01 et cr02, ainsi qu'en langage C++ sous IDE Arduino, uniquement pour le module CR02.

Pour le module RL01, les développements se sont arrêtés au stade de la tentative de configuration du transmetteur LoRa.

Le module CR02 a finalement été retenu pour le démonstrateur.

## 2.1. Module RL01

Le module RL01 étant composé d'un bridge I2C-SPI et d'un transmetteur LoRa, les documentations de ces deux composants ont été étudiées puis résumées pour faciliter les développements par la suite.

Afin de pouvoir configurer et utiliser le module RL01, la première étape consistait à configurer le bridge et à mettre au point des fonctionnalités de lecture et d'écriture vers son buffer.

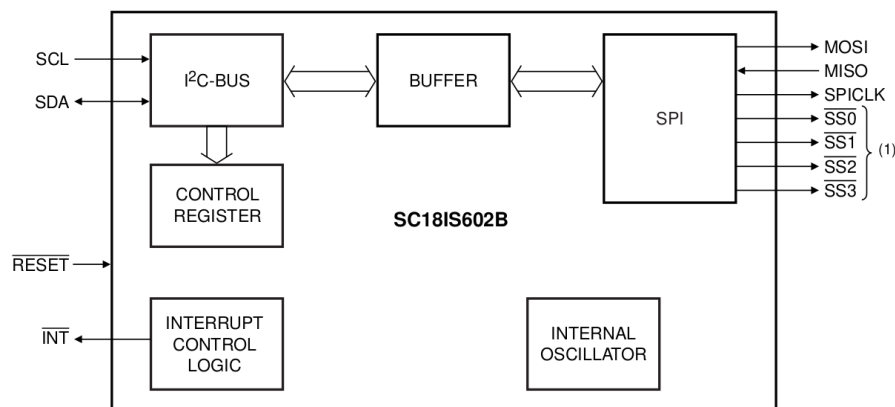


Illustration 7: Schéma bloc du bridge I2C-SPI

Les fonctions permettant l'initialisation du bridge, l'écriture et la lecture du buffer ont été écrites à partir des fonctions I2C du bsp.

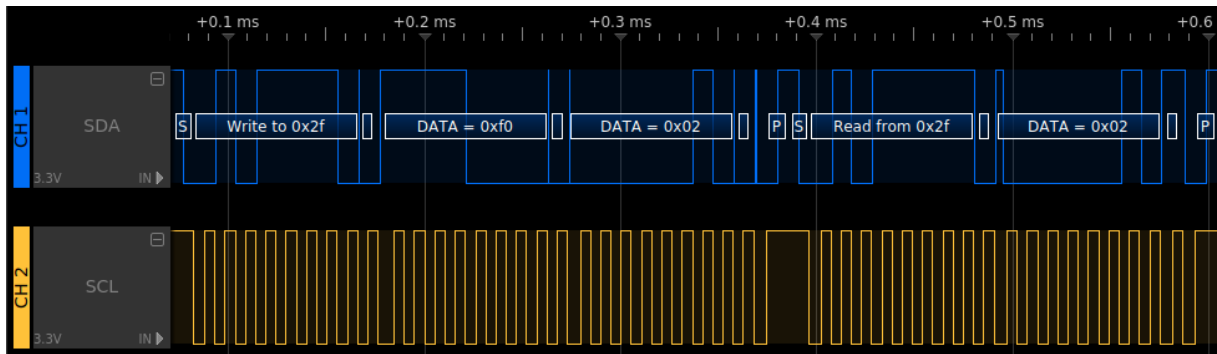


Illustration 8: Trames I2C d'écriture et de lecture du buffer du bridge

La capture de trames I2C a permis de confirmer le bon fonctionnement des écritures et lectures du buffer. L'illustration ci-dessus montre une écriture à l'adresse du buffer (0x2F), suivie d'une commande (0xF0) et de la donnée à écrire (0x02). Vient ensuite une lecture à l'adresse du buffer puis la donnée lue, identique à la donnée écrite.

Cette partie ayant été validée, il fallait ensuite pouvoir configurer le transmetteur LoRa.

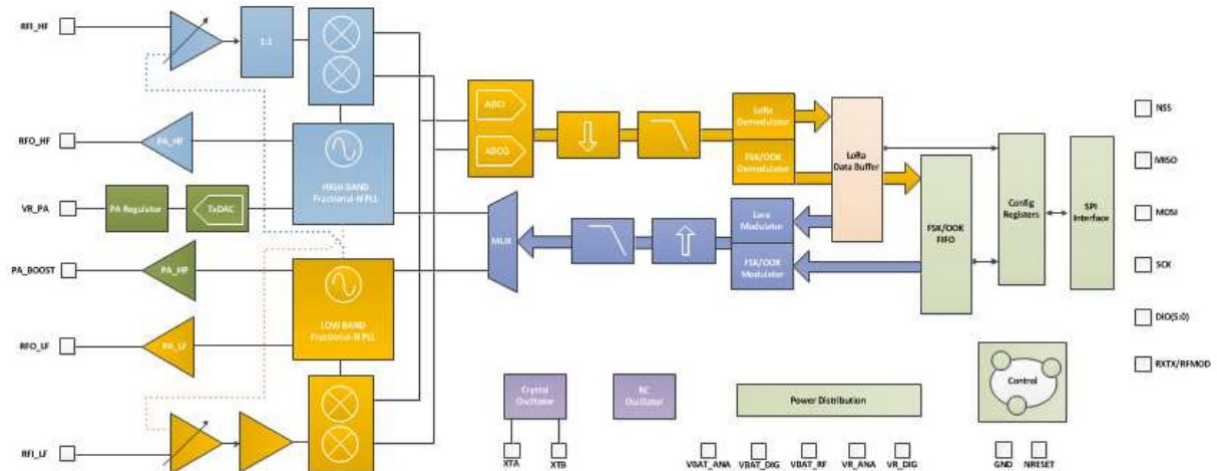


Illustration 9: Schéma bloc du transmetteur

Les tests ont consisté en l'écriture puis la lecture d'une valeur dans un des registres de configuration du transmetteur LoRa.

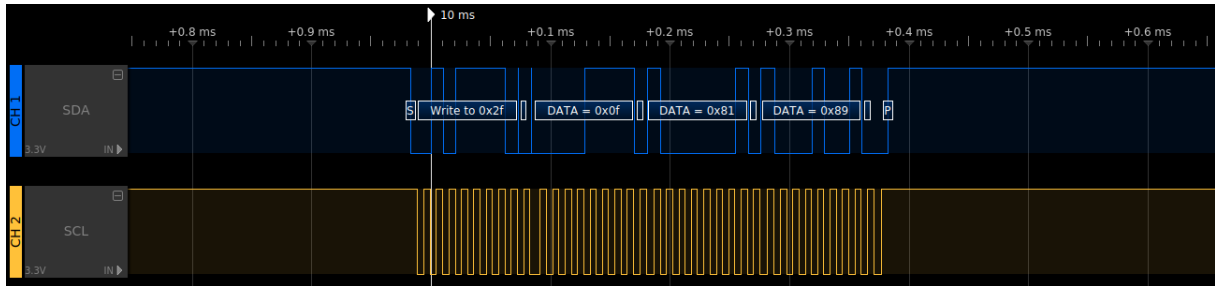


Illustration 10: Trames I2C d'écriture dans le registre RegOpMode du transmetteur

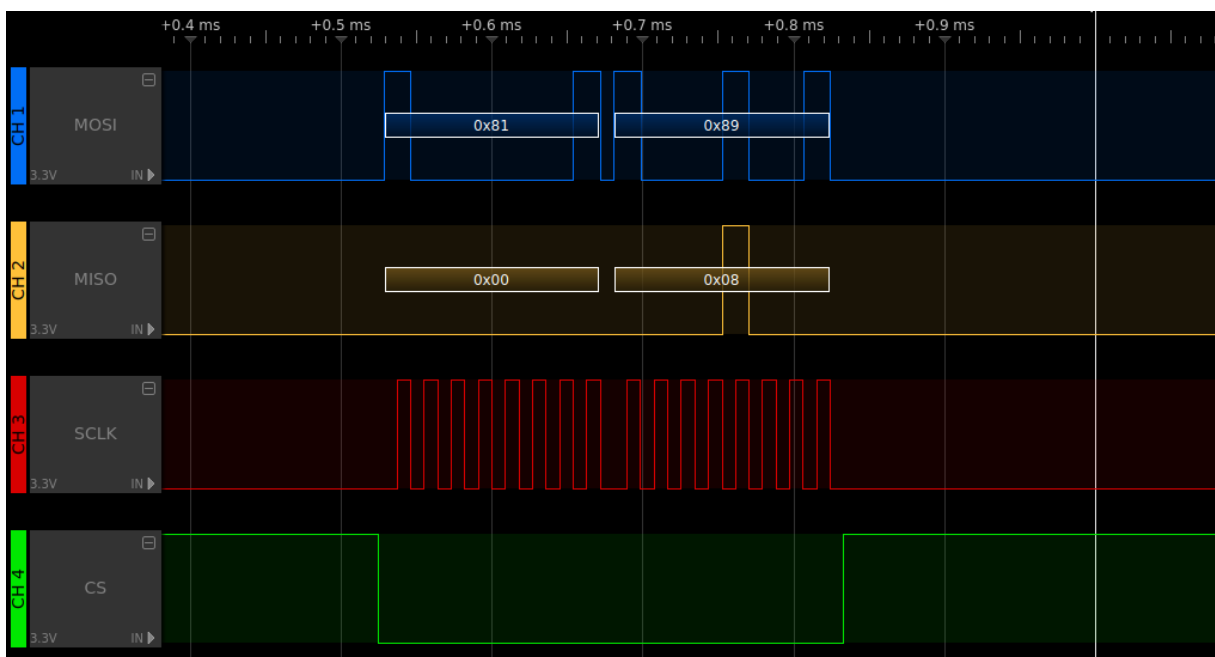


Illustration 11: Trames SPI d'écriture dans le registre RegOpMode du transmetteur

La trame I2C montre la demande d'écriture à l'adresse du bridge (0x2F), une commande d'écriture vers l'interface SPI (0x0F), l'adresse du registre en écriture (0x81 = registre 0x01 + écriture) et la donnée à écrire (0x89).

La trame SPI montre les données, sans le protocole I2C, arrivant au niveau du transmetteur.

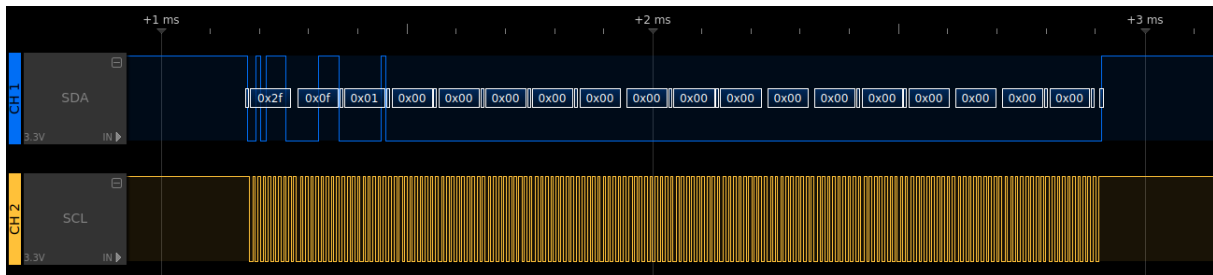


Illustration 12: Trames I2C de demande de lecture des registres

La trame I2C ci-dessus correspond à la demande de lecture du registre RegOpMode (0x01 = 0x01 + lecture).

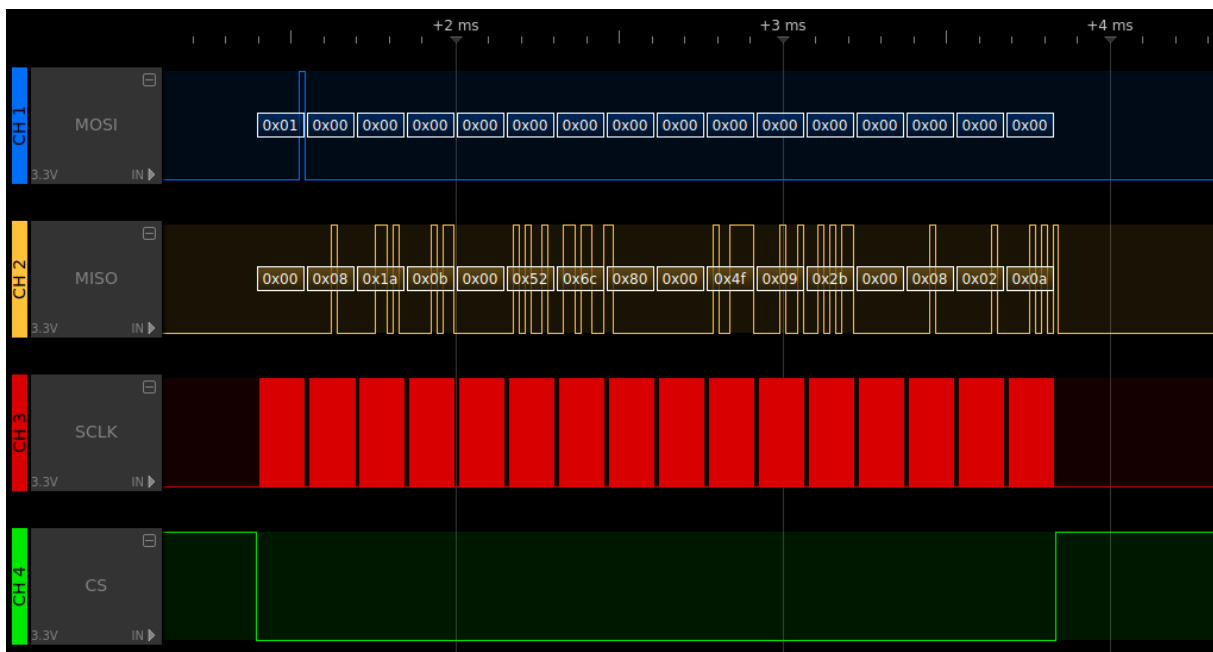


Illustration 13: Trames SPI de lecture des registres

La demande de lecture est transmise par l'intermédiaire du bridge. La trame SPI montre également les données retournées suite à cette demande.

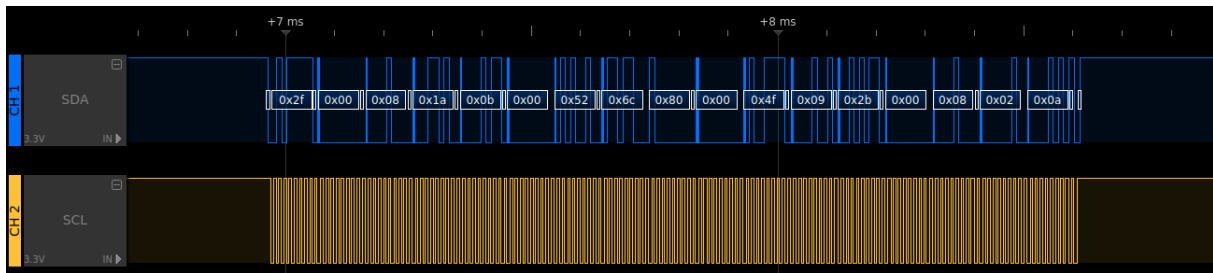


Illustration 14: Trames I2C de lecture des registres

Les données renvoyées sous protocole SPI, sont ensuite transmises par le bridge et converties au format I2C.

L'analyse des trames I2C et SPI de lecture des registres du transmetteur permet de mettre en évidence que l'écriture dans le registre n'a pas été effectuée. Le registre RegOpMode prend la valeur 0x08 correspondant à la transmission FSK en mode sleep.

Name (Address)	Bits	Variable Name	Mode	Default value	FSK/OOK Description
RegFifo (0x00)	7-0	Fifo	rw	0x00	FIFO data input/output
Registers for Common settings					
	7	LongRangeMode	r	0x00	0 → FSK/OOK Mode 1 → LoRa™ Mode This bit can be modified only in Sleep mode. A write operation on other device modes is ignored.
	6-5	ModulationType	rw	0x00	Modulation scheme: 00 → FSK 01 → OOK 10 → 11 → reserved
RegOpMode (0x01)	4	reserved	r	0x0	reserved
	3	LowFrequencyModeOn	rw	0x01	Access Low Frequency Mode registers (from address 0x61 on) 0 → High Frequency Mode (access to HF test registers) 1 → Low Frequency Mode (access to LF test registers)
	2-0	Mode	rw	0x01	Transceiver modes 000 → Sleep mode 001 → Stdby mode 010 → FS mode TX (FSTx) 011 → Transmitter mode (Tx) 100 → FS mode RX (FSRx) 101 → Receiver mode (Rx) 110 → reserved 111 → reserved
RegBitRateMsb (0x02)	7-0	BitRate(15:8)	rw	0x1a	MSB of Bit Rate (chip rate if Manchester encoding is enabled)
RegBitRateLsb (0x03)	7-0	BitRate(7:0)	rw	0x0b	LSB of bit rate (chip rate if Manchester encoding is enabled) $BitRate = \frac{FXOSC}{BitRate(15,0) + \frac{BitRateFrac}{16}}$ Default value: 4.8 kb/s
RegFdevMsb (0x04)	7-6	reserved	rw	0x00	reserved
	5-0	Fdev(13:8)	rw	0x00	MSB of the frequency deviation
RegFdevLsb (0x05)	7-0	Fdev(7:0)	rw	0x52	LSB of the frequency deviation $Fdev = Fstep \times Fdev(15,0)$ Default value: 5 kHz

Illustration 15: Première partie des registres du transmetteur en mode FSK

La comparaison des données retournées avec les valeurs des registres dans la documentation permet de constater qu'il s'agit des valeurs par défaut des registres lorsque le module se trouve en mode FSK.

Les développements réalisés sur le module RL01 n'ont pas permis de le configurer. La méthode d'écriture dans les registres ne fonctionne pas et est à revoir. La lecture des registres semble fonctionner pour une suite de registres mais ne permet pas de lire un seul registre comme voulu.

Il a finalement été décidé de poursuivre avec le module CR02.



## 2.2. Module CR02

### 2.2.1. Mesure et mise à disposition des données pour l'émetteur

Cette partie repose sur le travail déjà réalisé lors du projet immersif. Le projet cr02 reprend donc en grande partie le projet htU21d (capteur de température et d'humidité). Celui-ci a simplement été amélioré et augmenté de quelques instructions permettant l'envoi des données au module CR02. Celles-ci sont expédiées avec un intervalle de temps d'environ 3s.

Le PIC18 embarque un firmware réalisant :

- l'initialisation des communications (UART et I2C)
- la configuration du capteur de température et d'humidité
- la récupération des données du capteur
- l'envoi des données sur le bus I2C à destination du module CR02
- l'affichage en console pour les phases de développement et de debug

La donnée retournée par le capteur tient sur deux octets. L'architecture 8bits impose de transmettre octet par octet, la donnée est donc décomposée en deux parties pendant tout le processus de transmission et ne sera reconstituée qu'une fois réceptionnée au niveau de la Raspberry Pi.

## 2.2.2. Émission des données par communication LoRa

Le module CR02 étant muni d'un microcontrôleur ATmega328P, il a été possible de le programmer, sous environnement Arduino, au moyen de fonctions incluses dans les bibliothèques Wire, pour le protocole I2C, et RadioHead pour la communication LoRa, en s'inspirant des exemples `slave_receiver.ino` pour la lecture de données I2C et `CR02_client.ino` pour l'envoi de données par communication LoRa.

Le programme développé pour l'émetteur LoRa comporte dans un premier temps une phase d'initialisation permettant de configurer les communications UART et I2C ainsi que le transmetteur LoRa. La présence d'une donnée adressée au module CR02 sur le bus I2C déclenche un événement effectuant la lecture de deux octets. Les octets sont ensuite émis au travers d'une communication LoRa. Un affichage console est également possible pour les phases de développement et de debug.

```
~Curiosity-                               ~cr02_transmitter~
Sending to cr02_transmitter ...             Sending to cr02_receiver ...
0x655c -> Temperature : 22.72 °C           0x655C
0x73be -> Relative humidity : 50.51 %      ~cr02_transmitter~
                                           Sending to cr02_receiver ...
                                           0x73BE
~Curiosity-                               ~cr02_transmitter~
Sending to cr02_transmitter ...             Sending to cr02_receiver ...
0x6564 -> Temperature : 22.74 °C           0x6564
0x73b2 -> Relative humidity : 50.48 %      ~cr02_transmitter~
                                           Sending to cr02_receiver ...
                                           0x73B2
```

*Illustration 16: Affichage console des données au niveau du PIC18 et de l'émetteur LoRa*

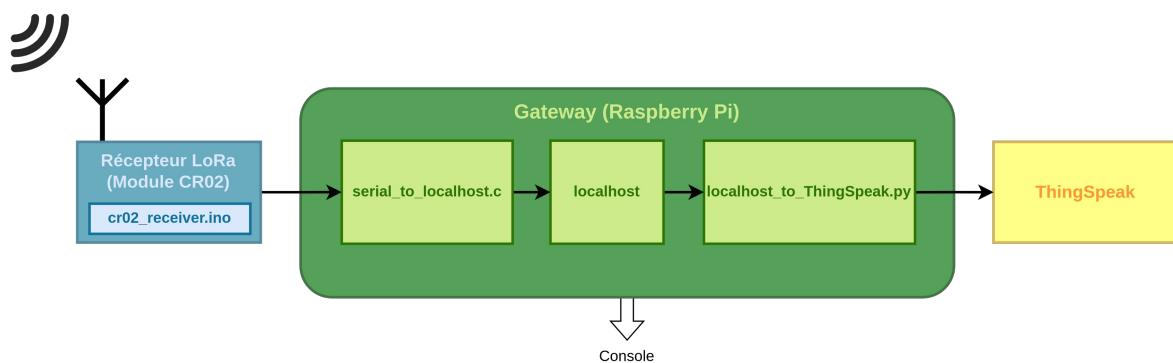
L'affichage en console à chaque étape de la transmission permet de vérifier la validité et la cohérence des données transmises.

# GATEWAY

---

*L'objectif de la partie « Gateway » est la réception des données transportées par protocole de communication wireless LoRa, ainsi que leur transmission à un serveur permettant de les rendre accessibles pour les interfaces utilisateur smartphone et PC.*

## 1. Applications de la gateway



*Illustration 17: Applications de la gateway*

Les deux missions de la gateway résident en la réception puis la transmission des données.

Pour la réception des données, le programme `cr02_receiver.ino` écrit pour le module LoRa récepteur permet la lecture des données transmises par communication LoRa puis leur écriture sur le port série. La Raspberry Pi effectue ensuite une lecture du port série, puis met en forme les données pour les préparer à leur transmission.

Pour la transmission des données, la Raspberry Pi dispose de deux applications, `serial_to_localhost.c` et `localhost_to_ThingSpeak.py`, permettant de transmettre les données en local dans un premier temps puis vers ThingSpeak dans un second temps.

## 2. Réception des données

### 2.1. Module CR02 récepteur

Les données sont réceptionnées, côté gateway, au moyen d'un module CR02 dont l'application a été développée à partir des fonctions provenant de la bibliothèque RadioHead. L'exemple CR02\_server.ino a servi de base et a été modifié de façon à lire les deux octets de donnée arrivant par communication LoRa, puis à les écrire sur le port série dans le bon ordre. Un affichage console est réalisé pour le debug.

```
~Curiosity~  
Sending to cr02_transmitter ...  
0x6598 -> Temperature : 22.88 °C  
0x5092 -> Relative humidity : 33.33 %  
  
~Curiosity~  
Sending to cr02_transmitter ...  
0x656c -> Temperature : 22.76 °C  
0x4fba -> Relative humidity : 32.92 %  
  
~cr02_receiver~  
Sending to serial ...  
0x6598 -> Temperature : 22.88 °C  
  
~cr02_receiver~  
Sending to serial ...  
0x5092 -> Relative humidity : 33.33 %  
  
~cr02_receiver~  
Sending to serial ...  
0x656C -> Temperature : 22.76 °C  
  
~cr02_receiver~  
Sending to serial ...  
0x4FBA -> Relative humidity : 32.92 %
```

*Illustration 18: Affichage console des données au niveau du PIC18 et du récepteur LoRa*

La majorité du temps, les données reçues par le récepteur LoRa sont identiques aux données d'origine en provenance du capteur.

```
~Curiosity~
Sending to cr02_transmitter ...
0x62f8 -> Temperature : 21.08 °C
0x5daa -> Relative humidity : 39.73 %

~Curiosity~
Sending to cr02_transmitter ...
0x62f4 -> Temperature : 21.07 °C
0x5746 -> Relative humidity : 36.61 %

~Curiosity~
Sending to cr02_transmitter ...
0x62fc -> Temperature : 21.09 °C
0x54ca -> Relative humidity : 35.39 %

~cr02_receiver~
Sending to serial ...
0x1A62 -> Relative humidity : 6.87 %

~cr02_receiver~
Sending to serial ...
0xF85D -> Temperature : 123.62 °C

~cr02_receiver~
Sending to serial ...
0xAA62 -> Relative humidity : 77.19 %

~cr02_receiver~
Sending to serial ...
0xF446 -> Relative humidity : 113.27 %

~cr02_receiver~
Sending to serial ...
0x62FC -> Temperature : 21.09 °C

~cr02_receiver~
Sending to serial ...
0x54CA -> Relative humidity : 35.39 %
```

*Illustration 19: Apparition de valeurs erronées lors de la réception des données*

Toutefois, un décalage peut parfois survenir lors de la réception des octets produisant ainsi des valeurs erronées se propageant ensuite dans le reste de la chaîne de transmission.

## 2.2. Lecture du port série

Les données arrivent sur le port série mais elles ne sont pas encore utilisables. Elles doivent encore être récupérées puis traitées avant de pouvoir être transmises.

Le programme écrit en langage C permettant la lecture du port série repose sur l'utilisation d'une structure `termios` configurée en mode non canonique de façon à récupérer le nombre d'octets voulu.

Dans un premier temps, l'application a été testée en prenant pour donnée une chaîne de caractères, écrite sur le port série toutes les 3s.

```
lucile@pc:~/Documents/projet_industriel/gateway$ ./read_uart
DUB
DUB
DUB
```

*Illustration 20: Lecture d'une chaîne de caractères sur le port série*

Le programme a ensuite été modifié pour lire les deux octets de donnée arrivant toutes les 3s en provenance du récepteur LoRa. Les deux octets de donnée lus sur le port série sont assemblés de façon à reconstituer la donnée du capteur. La valeur physique correspondant à la donnée est ensuite calculée. Un affichage de cette valeur physique dans un terminal permet de vérifier le bon fonctionnement du programme.

```
0x5c8e -> Relative humidity : 39.18 %
~Curiosity~
Sending to cr02_transmitter ...
0x624 -> Temperature : 20.42 °C
0x5cf2 -> Relative humidity : 39.37 %
~Curiosity~
Sending to cr02_transmitter ...
0x624 -> Temperature : 20.42 °C
0x5cee -> Relative humidity : 39.37 %
~Curiosity~
Sending to cr02_transmitter ...
0x6210 -> Temperature : 20.46 °C
0x5ce2 -> Relative humidity : 39.34 %

lucile@pc:~/Documents/projet_industriel/gateway$ ./read_uart
0x5c8e
Relative humidity : 39.18 %
0x6204
Temperature : 20.42 °C
0x5cf2
Relative humidity : 39.37 %
0x6204
Temperature : 20.42 °C
0x5cee
Relative humidity : 39.37 %
0x6210
Temperature : 20.46 °C
0x5ce2
Relative humidity : 39.34 %
█
```

*Illustration 21: Lecture des données du module CR02 sur le port série*

Les données après lecture du port série correspondent bien aux données de départ, elles ont été correctement lues puis traitées. Elles sont maintenant prêtes pour être transmises.

## 3. Transmission des données vers un broker

### 3.1. Protocole MQTT

MQTT (MQ Telemetry Transport) est un protocole de transport de messages basé sur TCP/IP, dont le principe de fonctionnement repose sur les mécanismes de publication et d'abonnement à des topics entre des clients et un serveur. Il s'agit d'un protocole léger, adapté à une utilisation dans des environnements contraints comme l'Internet des objets où l'empreinte du code doit être faible et lorsque les ressources en énergie et bande passante sont limitées.

Le modèle de publish/subscribe offre un découplage spatial entre les différents clients, ceux-ci ne se contactent jamais directement. L'intermédiaire entre les clients se fait au moyen d'un broker.

Les clients s'abonnent à un ou plusieurs topics. Un client peut publier des messages sur un topic choisi à destination d'un broker. Le rôle du broker est de filtrer tous les messages entrant, en fonction des topics, et de les distribuer correctement aux abonnés.

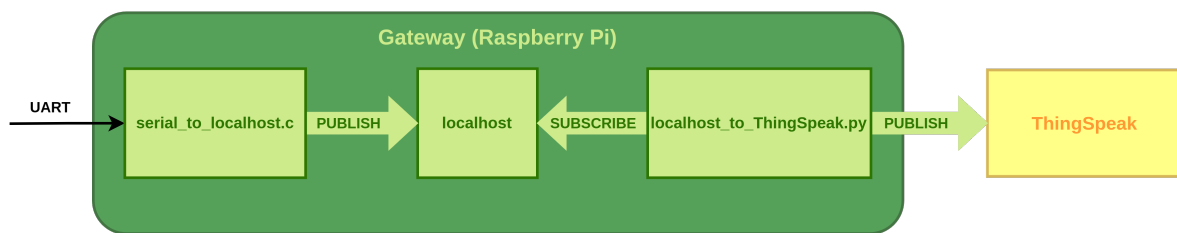
### 3.2. ThingSpeak

Dans le cadre du projet, ThingSpeak est utilisé pour vérifier le bon fonctionnement du programme `localhost_to_ThingSpeak.py` (envoi de messages par protocole MQTT vers un broker). ThingSpeak permet également de mettre les données à disposition pour les interfaces utilisateur.

Le channel ThingSpeak est configuré de manière à disposer d'un champ pour chaque type de donnée.

La création d'un device MQTT permet d'obtenir les identifiants nécessaires pour la configuration du client MQTT `localhost_to_ThingSpeak.py`.

### 3.3. Transmission des données en deux temps



*Illustration 22: Décomposition de la transmission*

La décomposition de la transmission en deux étapes est la conséquence de difficultés rencontrées pour développer un programme en langage C transmettant directement les données vers ThingSpeak. La solution de contournement a consisté en l'écriture de deux programmes au moyen de fonctions fournies par les bibliothèques MQTT Paho C et Paho python, permettant ainsi d'envoyer finalement la donnée vers ThingSpeak.

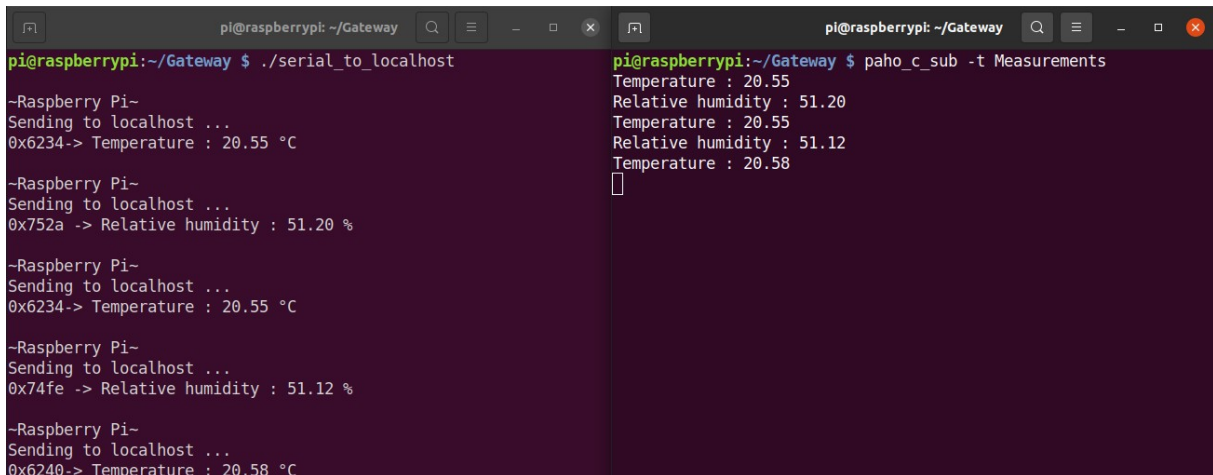
Les développements réalisés pour la transmission via protocole MQTT sont le résultat d'une collaboration avec Briac Panchot et Thomas Boduch.

#### 3.3.1. Première étape : serial\_to\_localhost.c

Cette première étape reprend le travail effectué précédemment pour la lecture du port série et y ajoute l'envoi de messages par protocole MQTT vers localhost. Les fonctions mises à disposition par la bibliothèque Paho C permettent de créer un client MQTT, de le configurer, puis de publier des messages sur un topic. L'exemple MQTTClient\_publish.c fourni avec la bibliothèque Paho C a servi de base pour développer ce programme.

Dans l'objectif de les rendre exploitables, les messages sont des chaînes de caractères mises sous la forme « type de la mesure : valeur ».





```
pi@raspberrypi:~/Gateway $ ./serial_to_localhost
~Raspberry Pi~
Sending to localhost ...
0x6234-> Temperature : 20.55 °C

~Raspberry Pi~
Sending to localhost ...
0x752a -> Relative humidity : 51.20 %

~Raspberry Pi~
Sending to localhost ...
0x6234-> Temperature : 20.55 °C

~Raspberry Pi~
Sending to localhost ...
0x74fe -> Relative humidity : 51.12 %

~Raspberry Pi~
Sending to localhost ...
0x6240-> Temperature : 20.58 °C

pi@raspberrypi:~/Gateway $ paho_c_sub -t Measurements
Temperature : 20.55
Relative humidity : 51.20
Temperature : 20.55
Relative humidity : 51.12
Temperature : 20.58
[]
```

*Illustration 23: Publication des messages vers localhost*

Le programme est testé au moyen de la ligne de commande `paho_c_sub` permettant de s'abonner au topic « Measurements » sur lequel les messages sont publiés par le client MQTT. Il est ainsi possible de vérifier que les messages ont été correctement publiés vers localhost.

### 3.3.2. Deuxième étape : `localhost_to_ThingSpeak.py`

Ce second programme réalisé au moyen de la bibliothèque Paho python permet dans un premier temps de souscrire au topic local « Measurements » de façon à récupérer les messages qui y sont publiés par le premier programme, ceux-ci sont ensuite publiés vers ThingSpeak.

Les messages reçus dans le format « type de la mesure : valeur » sont décodés de façon à déterminer le type de la donnée pour l'envoyer dans le champ correspondant du channel ThingSpeak.

Les messages sont publiés vers ThingSpeak sur le topic « channels/channel\_ID/publish » sous la forme « fieldX=valeur ».

```
~Raspberry Pi~
Sending to localhost ...
0x655c-> Temperature : 22.72 °C

~Raspberry Pi~
Sending to localhost ...
0x73be -> Relative humidity : 50.51 %

~Raspberry Pi~
Sending to localhost ...
0x6564-> Temperature : 22.74 °C

~Raspberry Pi~
Sending to localhost ...
0x73b2 -> Relative humidity : 50.48 %

~Raspberry Pi~
Sending to ThingSpeak ...
Relative humidity : 50.51

~Raspberry Pi~
Sending to ThingSpeak ...
Temperature : 22.74

~Raspberry Pi~
Sending to ThingSpeak ...
Relative humidity : 50.48
```

Illustration 24: Publication des messages vers ThingSpeak

L’affichage en console permet de vérifier que les données récupérées au niveau du localhost par le second programme sont identiques à celles publiées par le premier programme.

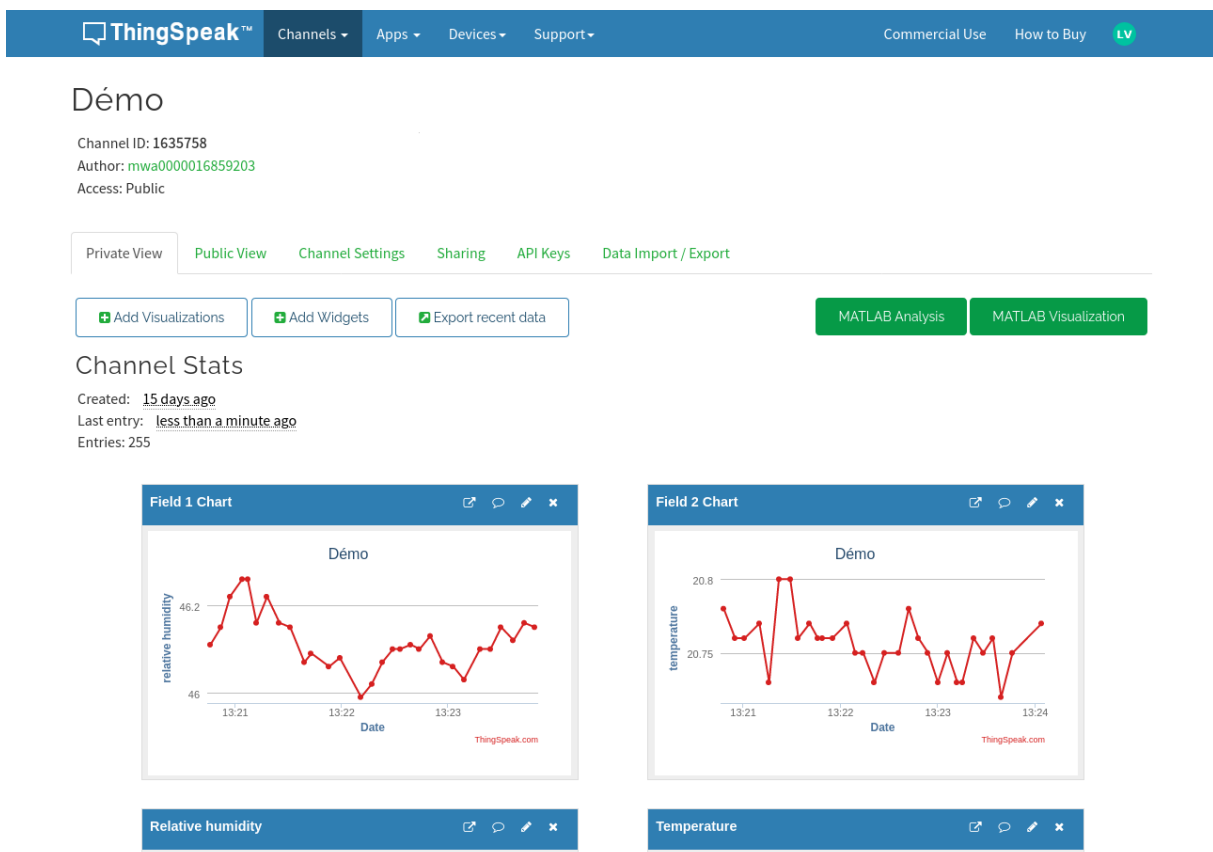


Illustration 25: Visualisation des données sous ThingSpeak

La visualisation des données sous ThingSpeak permet de vérifier que celles-ci ont été correctement transmises et récupérées dans les bons champs.

# CONCLUSION

---

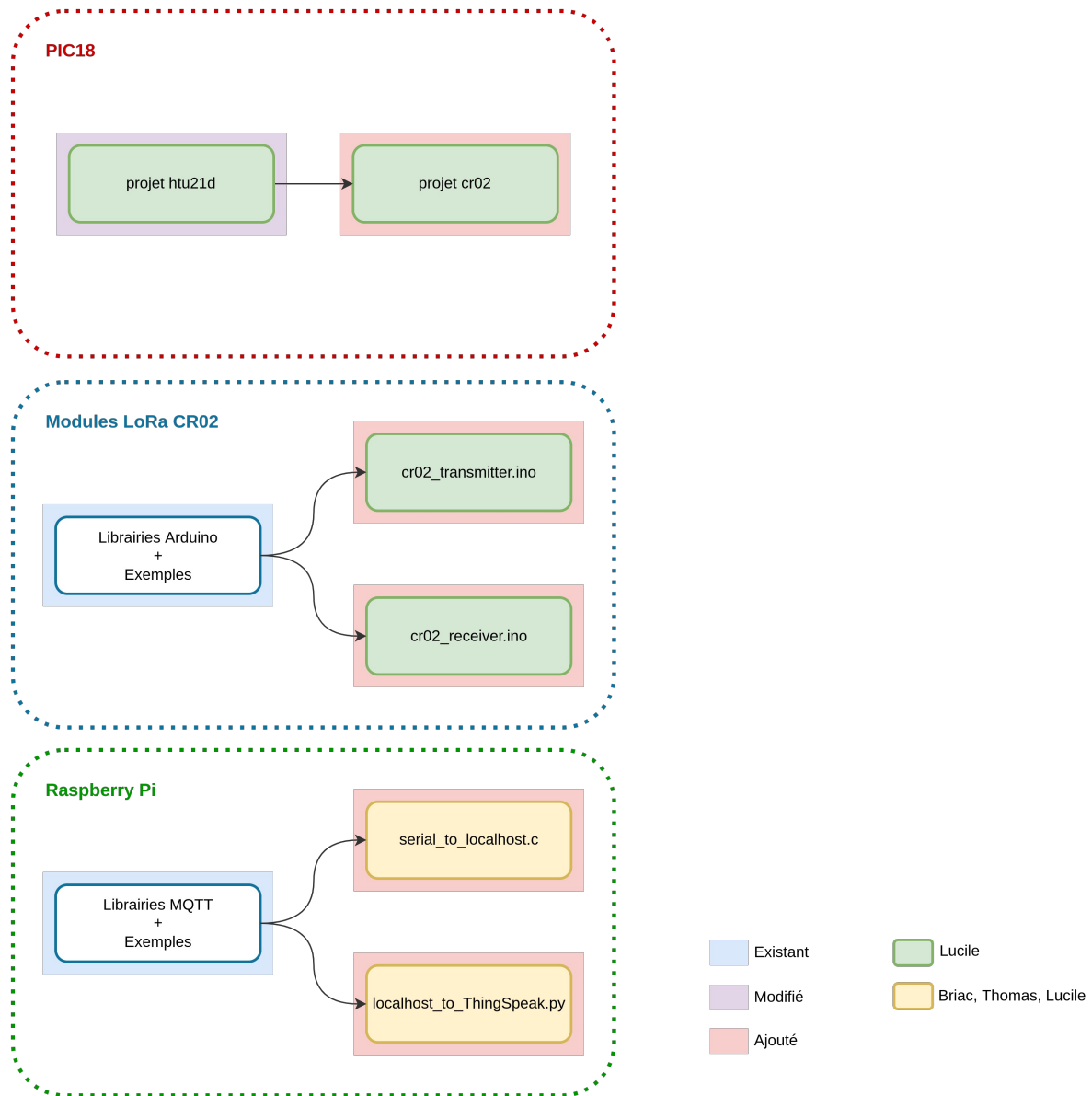


Illustration 26: Développements réalisés

Les développements réalisés ont permis d'obtenir un démonstrateur effectuant la transmission des données du capteur vers ThingSpeak. Celui-ci pourra servir de base de travail pour la prochaine génération d'ingénieurs qui travaillera sur le projet « I am Ensicaen ».

# ANNEXES

```
1  /*
2  * test program for cr02
3  * author
4  */
5
6  #include <stdio.h>
7  #include <bsplib.h>
8  #include <htu21d.h>
9  #include <cr02.h>
10
11 #define DELAY_3S() for (int i = 0; i < 15; i++) delay_200ms()
12
13 #define DEBUG_CONSOLE 1
14
15 void main(void)
16 {
17     uInt16_t temperature;
18     uInt16_t humidity;
19     uInt8_t checksum;
20
21 #if DEBUG_CONSOLE
22     float humidity_physical_value;
23     float temperature_physical_value;
24     uInt8_t tmp[50];
25 #endif
26
27     uInt8_t temperature_msb;
28     uInt8_t temperature_lsb;
29     uInt8_t humidity_msb;
30     uInt8_t humidity_lsb;
31
32     /* system init */
33     i2c1_init( I2C1_BAUDRATE );
34 #if DEBUG_CONSOLE
35     uart1_init( UART1_BAUD_RATE );
36 #endif
37
38     /* soft reset */
39     soft_reset( HTU21D_ADD );
40
41     /* choice of measurement resolution */
42     write_register( HTU21D_ADD, MEASUREMENT_RESOLUTION_11_11 );
43
44     while (1) {
45         read_measurement( HTU21D_ADD, TEMPERATURE_MEASUREMENT_HOLD_MASTER, &temperature, &checksum );
46         delay_10us();
47         read_measurement( HTU21D_ADD, HUMIDITY_MEASUREMENT_HOLD_MASTER, &humidity, &checksum );
48         delay_10us();
49
50         temperature_msb = (uInt8_t) (temperature >> 8) & 0x00FF;
51         temperature_lsb = (uInt8_t) temperature & 0x00FF;
52
53         humidity_msb = (uInt8_t) (humidity >> 8) & 0x00FF;
54         humidity_lsb = (uInt8_t) humidity & 0x00FF;
55
56 #if DEBUG_CONSOLE
57         humidity &= 0xfffc;
58         temperature_physical_value = ( ( temperature * 175.72 ) / 65536 ) - 46.85;
59         humidity_physical_value = ( ( humidity * 125.0 ) / 65536 ) - 6;
60
61         uart1_puts( "\n-Curiosity~\n\r" );
62         uart1_puts( "Sending to cr02_transmitter ... \n\n\r" );
63
64         sprintf( tmp, "0x%x%x", temperature_msb, temperature_lsb );
65         uart1_puts( tmp );
66         sprintf( tmp,
67                 " -> Temperature : %d.%02u °C",
68                 (uInt8_t) temperature_physical_value,
69                 (uInt8_t) ((temperature_physical_value - (uInt8_t) temperature_physical_value) * 100) );
70         uart1_puts( tmp );
71         uart1_puts( "\n\n\r" );
72
73         sprintf( tmp, "0x%x%x", humidity_msb, humidity_lsb );
74         uart1_puts( tmp );
75         sprintf( tmp,
76                 " -> Relative humidity : %d.%02u %%",
77                 (uInt8_t) humidity_physical_value,
78                 (uInt8_t) ((humidity_physical_value - (uInt8_t) humidity_physical_value) * 100) );
79         uart1_puts( tmp );
80         uart1_puts( "\n\n\r" );
81 #endif
82
83         i2c1_write(CR02_ADD, temperature_msb, temperature_lsb);
84
85         DELAY_3S();
86
87         i2c1_write(CR02_ADD, humidity_msb, humidity_lsb);
88
89         DELAY_3S();
90     }
91 }
```

projet cr02

```

slave_receiver
1 // Wire Slave Receiver
2 // by Nicholas Zambetti <http://www.zambetti.com>
3
4 // Demonstrates use of the Wire library
5 // Receives data as an I2C/TWI slave device
6 // Refer to the "Wire Master/Writer" example for use with this
7 // Created 29 March 2006
8
9 // This example code is in the public domain.
10
11 #include <Wire.h>
12
13 void setup() {
14   Wire.begin(8); // join I2C bus with address #8
15   Wire.onReceive(receiveEvent); // register event
16   Serial.begin(9600); // start serial for output
17 }
18
19 void loop() {
20   delay(100);
21 }
22
23 // function that executes whenever data is received from master
24 // this function is registered as an event, see setup()
25 void receiveEvent(int howMany) {
26   while (1 < Wire.available()) { // loop through all but the last
27     char c = Wire.read(); // receive byte as a character
28     Serial.print(c); // print the character
29   }
30   int x = Wire.read(); // receive byte as an integer
31   Serial.println(x); // print the integer
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

## cr02\_transmitter.ino avec exemples ayant servi de base

CR02_server	cr02_receiver
<pre> 1 #include &lt;RH_RF95.h&gt; 2 3 #define LED_BUILTIN 16 4 5 #define CR02_FREQUENCY 868.0 6 7 uint8_t tempdata[30]; 8 9 /* 10  * Slave Select mapped to D13/PD2 11  * Interrupt mapped to D14/PB2 12  */ 13 RH_RF95 CR02; 14 15 void setup() 16 { 17   pinMode(LED_BUILTIN, OUTPUT); 18 19   Serial.begin(115200); 20 21   if (!CR02.init()) { 22     Serial.println("init failed"); 23   } 24   // The default transmitter power is 13dBm, using PA_BOOST. 25   // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then 26   // you can set transmitter powers from 5 to 23 dBm: 27   // Failure to do that will result in extremely low transmit powers. 28 29   //CR02.setModemConfig(CR02.Bw31_25Cr48Sf512); 30   CR02.setFrequency(CR02_FREQUENCY); 31   CR02.setTxPower(23, false); 32 } 33 34 void loop() 35 { 36   if (CR02.available()) 37   { 38     // Should be a message for us now 39     uint8_t buf[RH_RF95_MAX_MESSAGE_LEN]; 40     uint8_t len = sizeof(buf); 41     if (CR02.recv(buf, &amp;len)) 42     { 43       digitalWrite(LED_BUILTIN, HIGH); 44       Serial.print("got request: "); 45       Serial.println((char*)buf); 46       Serial.print("RSSI: "); 47       Serial.println(CR02.LastRssi(), DEC); 48 49       // Send a reply 50       sprintf(tempdata, "%s, Hello Client"); 51       CR02.send(tempdata, sizeof(tempdata)); 52       CR02.waitPacketSent(); 53       Serial.println("Sent a reply"); 54       digitalWrite(LED_BUILTIN, LOW); 55     } 56     else 57     { 58       Serial.println("recv failed"); 59     } 60   } 61 } </pre>	<pre> 1 #include &lt;RH_RF95.h&gt; 2 3 #define LED_BUILTIN 16 4 5 #define CR02_FREQUENCY 868.0 6 7 #define TRUE 1 8 #define FALSE 0 9 10 #define DEBUG_CONSOLE 0 11 12 uint8_t payload; 13 uint8_t payload_msb; 14 uint8_t payload_lsb; 15 uint8_t len = sizeof(payload); 16 17 uint8_t is_msb = TRUE; 18 uint8_t nb_byte = 0; 19 20 char buf[50]; 21 uint8_t status_measurement; 22 uint16_t data; 23 float physical_value; 24 25 /* 26  * Slave Select mapped to D13/PD2 27  * Interrupt mapped to D14/PB2 28  */ 29 RH_RF95 CR02; 30 31 void setup() 32 { 33   pinMode(LED_BUILTIN, OUTPUT); 34 35   Serial.begin(9600, SERIAL_8N1); // start serial for output 36 37   if (!CR02.init()) { 38     #if DEBUG_CONSOLE 39       Serial.println("init failed"); 40     #endif 41   } 42 43   // The default transmitter power is 13dBm, using PA_BOOST. 44   // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then 45   // you can set transmitter powers from 5 to 23 dBm: 46   // Failure to do that will result in extremely low transmit powers. 47 48   //CR02.setModemConfig(CR02.Bw31_25Cr48Sf512); 49   CR02.setFrequency(CR02_FREQUENCY); 50   CR02.setTxPower(23, false); 51 } 52 53 void loop() 54 { 55   if (CR02.available()) { 56     if (CR02.recv(&amp;payload, &amp;len)) { 57       digitalWrite(LED_BUILTIN, HIGH); 58 59       if (is_msb) { 60         payload_msb = payload; 61         is_msb = FALSE; 62         nb_byte++; 63       } else { 64         payload_lsb = payload; 65         is_msb = TRUE; 66         nb_byte++; 67       } 68 69       digitalWrite(LED_BUILTIN, LOW); 70 71     } else { 72       #if DEBUG_CONSOLE 73         Serial.println("recv failed"); 74       #endif 75     } 76 77     if (nb_byte == 2) { 78 79       #if DEBUG_CONSOLE 80         data = payload_msb &lt;&lt; 8; 81         data  = payload_lsb; 82 83         Serial.println("\n-cr02_receiver-"); 84         Serial.println("Sending to serial ..."); 85         Serial.print("0x"); 86         Serial.print(payload_msb, HEX); 87         Serial.print(payload_lsb, HEX); 88 89         status_measurement = payload_lsb &amp; 0x02; 90         if (status_measurement == 0x02) { 91           data &amp;= 0xffff; 92           physical_value = ((data * 125.0) / 65536) - 6; 93           Serial.print("-&gt; Relative humidity : "); 94           sprintf(buf, 95                 "%d.%02u", 96                 (uint8_t) physical_value, 97                 (uint8_t) ((physical_value - (uint8_t) physical_value) * 100)); 98           Serial.print(buf); 99           Serial.println(" %"); 100         } 101         if (status_measurement == 0x00) { 102           physical_value = ((data * 175.72) / 65536) - 46.85; 103           Serial.print("-&gt; Temperature : "); 104           sprintf(buf, 105                 "%d.%02u", 106                 (uint8_t) physical_value, 107                 (uint8_t) ((physical_value - (uint8_t) physical_value) * 100)); 108           Serial.print(buf); 109           Serial.println(" °C"); 110         } 111       } 112     } 113   } 114 115   #if DEBUG_CONSOLE 116     Serial.write(payload_lsb); 117     Serial.write(payload_msb); 118   #endif 119 120   nb_byte = 0; 121 } 122 } 123 } 124 } </pre>

*cr02\_receiver.ino avec exemple ayant servi de base*

```

C MQTTClient_publish.c X
1  /*-----*/
2  * Copyright (c) 2012, 2020 IBM Corp.
3  *
4  * All rights reserved. This program and the accompanying materials
5  * are made available under the terms of the Eclipse Public License v2.0
6  * and Eclipse Distribution License v1.0 which accompany this distribution.
7  *
8  * The Eclipse Public License is available at
9  * https://www.eclipse.org/legal/epl-2.0/
10 * and the Eclipse Distribution License is available at
11 * http://www.eclipse.org/org/documents/edl-v10.php.
12 *
13 * Contributors:
14 *   Ian Craggs - initial contribution
15 *-----*/
16
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include "MQTTClient.h"
21
22 #define ADDRESS "tcp://mqtt.eclipse.org:1883"
23 #define CLIENTID "ExampleClientPub"
24 #define TOPIC "MQTT Examples"
25 #define PAYLOAD "Hello World!"
26 #define QOS 1
27 #define TIMEOUT 1000UL
28
29 int main(int argc, char* argv[])
30 {
31     MQTTClient client;
32     MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
33     MQTTClient_message pubmsg = MQTTClient_message_initializer;
34     MQTTClient_deliveryToken token;
35     int rc;
36
37     if ((rc = MQTTClient_create(&client, ADDRESS, CLIENTID,
38         MQTTCLIENT_PERSISTENCE_NONE, NULL)) != MQTTCLIENT_SUCCESS)
39     {
40         printf("Failed to create client, return code %d\n", rc);
41         exit(EXIT_FAILURE);
42     }
43
44     conn_opts.keepAliveInterval = 20;
45     conn_opts.cleansession = 1;
46     if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
47     {
48         printf("Failed to connect, return code %d\n", rc);
49         exit(EXIT_FAILURE);
50     }
51
52     pubmsg.payload = PAYLOAD;
53     pubmsg.payloadlen = (int)strlen(PAYLOAD);
54     pubmsg.qos = QOS;
55     pubmsg.retain = 0;
56     if ((rc = MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token)) != MQTTCLIENT_SUCCESS)
57     {
58         printf("Failed to publish message, return code %d\n", rc);
59         exit(EXIT_FAILURE);
60     }
61
62     printf("Waiting for up to %d seconds for publication of %s\n",
63         (int)TIMEOUT/1000, PAYLOAD, CLIENTID);
64     rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
65     printf("Message with delivery token %d delivered\n", token);
66
67     if ((rc = MQTTClient_disconnect(client, 1888)) != MQTTCLIENT_SUCCESS)
68     {
69         printf("Failed to disconnect, return code %d\n", rc);
70     }
71     MQTTClient_destroy(&client);
72     return rc;
73 }
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
}

C serial_to_localhost.c X
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <termios.h>
5 #include <fcntl.h>
6 #include <errno.h>
7 #include <string.h>
8
9 #include "MQTTClient.h"
10
11 #define ADDRESS "localhost:1883"
12 #define CLIENTID "ClientPubLocalhost"
13 #define TOPIC "Measurements"
14 #define QOS 1
15 #define TIMEOUT 6000L
16
17 #define DEBUG_CONSOLE 1
18
19 int main()
20 {
21     int serial_port = open("/dev/ttyUSB0", O_RDONLY);
22     if (serial_port == -1) {
23         printf("Error %i from open: %s\n", errno, strerror(errno));
24         exit(EXIT_FAILURE);
25     }
26
27     struct termios tty_options;
28
29     if (tcgetattr(serial_port, &tty_options) != 0) {
30         printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
31         exit(EXIT_FAILURE);
32     }
33
34     tty_options.c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
35     tty_options.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
36     tty_options.c_cflag &= ~CSIZE; // Clear all bits that set the data size
37     tty_options.c_cflag |= CS8; // 8 bits per byte (most common)
38     tty_options.c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
39     tty_options.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
40
41     tty_options.c_lflag &= ~ICANON;
42     tty_options.c_lflag &= ~ECHO; // Disable echo
43     tty_options.c_lflag &= ~ECHOE; // Disable erasure
44     tty_options.c_lflag &= ~ECHONL; // Disable new-line echo
45     tty_options.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
46     tty_options.c_lflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl.
47     tty_options.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL); // Disable any special handling of received bytes
48     tty_options.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
49     tty_options.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
50     // tty_options.c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)
51     // tty_options.c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT ON LINUX)
52
53     tty_options.cc[VTIME] = 1;
54     tty_options.cc[VMIN] = 2;
55
56     cfsetispeed(&tty_options, B9600);
57     cfsetospeed(&tty_options, B9600);
58
59     if (tcsetattr(serial_port, TCSANOW, &tty_options) != 0) {
60         printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
61         exit(EXIT_FAILURE);
62     }
63
64     MQTTClient client;
65     MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
66     MQTTClient_message pubmsg = MQTTClient_message_initializer;
67     MQTTClient_deliveryToken token;
68     int rc;
69
70     if ((rc = MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL)) != MQTTCLIENT_SUCCESS) {
71         printf("Failed to create client, return code %d\n", rc);
72         exit(EXIT_FAILURE);
73     }
74
75     conn_opts.keepAliveInterval = 20;
76     conn_opts.cleansession = 1;
77     if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
78         printf("Failed to connect, return code %d\n", rc);
79         exit(EXIT_FAILURE);
80     }
81
82     u_int16_t payload;
83     u_int8_t measurement_id;
84     float humidity_physical_value;
85     float temperature_physical_value;
86
87     char tmp[50];
88     int num_bytes = 0;
89
90     while (1) {
91         num_bytes = read(serial_port, &payload, sizeof(payload));
92
93         if (num_bytes < 0) {
94             printf("Error reading: %s", strerror(errno));
95             exit(EXIT_FAILURE);
96         }
97
98         #if DEBUG_CONSOLE
99             printf("\n-Raspberry Pi-\n");
100             printf("Sending to localhost...\n");
101             printf("0x%x", payload);
102         #endif
103
104         measurement_id = payload & 0x02;
105
106         if (measurement_id == 0x02) {
107             payload &= 0xFFFC;
108             humidity_physical_value = ((payload * 125.0) / 65536) - 6;
109             sprintf(tmp, "Relative humidity: %d.802u",
110                 (u_int8_t) humidity_physical_value,
111                 (u_int8_t) ((humidity_physical_value - (u_int8_t) humidity_physical_value) * 100));
112             #if DEBUG_CONSOLE
113                 printf("-> %s %d\n", tmp);
114             #endif
115         }
116         if (measurement_id == 0x00) {
117             temperature_physical_value = ((payload * 175.72) / 65536) - 46.85;
118             sprintf(tmp, "Temperature: %d.802u",
119                 (u_int8_t) temperature_physical_value,
120                 (u_int8_t) ((temperature_physical_value - (u_int8_t) temperature_physical_value) * 100));
121             #if DEBUG_CONSOLE
122                 printf("-> %s %d\n", tmp);
123             #endif
124         }
125
126         pubmsg.payload = tmp;
127         pubmsg.payloadlen = (int)strlen(tmp);
128         pubmsg.qos = QOS;
129         pubmsg.retain = 0;
130         if ((rc = MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token)) != MQTTCLIENT_SUCCESS) {
131             printf("Failed to publish message, return code %d\n", rc);
132             exit(EXIT_FAILURE);
133         }
134         rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
135     }
136     return EXIT_SUCCESS;
137 }

```

*serial\_to\_localhost.c avec exemple ayant servi de base*

```

localhost_to_ThingSpeak.py x
1  #!/usr/bin/env python3
2  # 20/01/2022
3  # Publish data with mqtt to a ThingSpeak server
4  # OPEN-SOURCE
5
6  import paho.mqtt.publish as publish
7  import paho.mqtt.client as mqtt
8  import string
9  import socket
10 import time
11
12 #####
13 # on_message #
14 #####
15 def on_message(client,userdata, msg):
16     message_received=str(msg.payload.decode("utf-8"))
17     channel_ID = "1635758"
18     t_port = 80
19     t_transport="websockets"
20     mqtt_host = "mqtt3.thingspeak.com"
21     mqtt_client_ID = "ORYNAwQDASUICCE9IAotHi8"
22     mqtt_username = "ORYNAwQDASUICCE9IAotHi8"
23     mqtt_password = "Uew5ZBCRq0jæe84GAbjrN/ad"
24     topic_ = "channels/" + channel_ID + "/publish"
25     data_type=message_received.split(" :")[0]
26     data=message_received.split(" : ")[1]
27     if(data_type == "Relative humidity"):
28         payload="field1="+data
29     elif(data_type == "Temperature"):
30         payload="field2="+data
31
32     try:
33         print("\n-Raspberry Pi~")
34         print("Sending to ThingSpeak ...")
35         print(data_type, " : ", data, "\n")
36         publish.single(topic_, payload, hostname=mqtt_host, transport=t_transport,
37                       port=t_port, client_id=mqtt_client_ID, auth={'username':mqtt_username,'password':mqtt_password})
38     except Exception as e:
39         print(e)
40
41
42 if __name__ == '__main__':
43     topic = "Measurements"
44     client = mqtt.Client("test")
45     client.on_message = on_message
46     client.connect("localhost",1883,60)
47     client.subscribe(topic)
48
49     try:
50         client.loop_forever()
51     except KeyboardInterrupt:
52         client.disconnect()
53         client.loop_stop()
54

```

*localhost\_to\_ThingSpeak.py*





## École Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053  
14050 CAEN cedex 04

