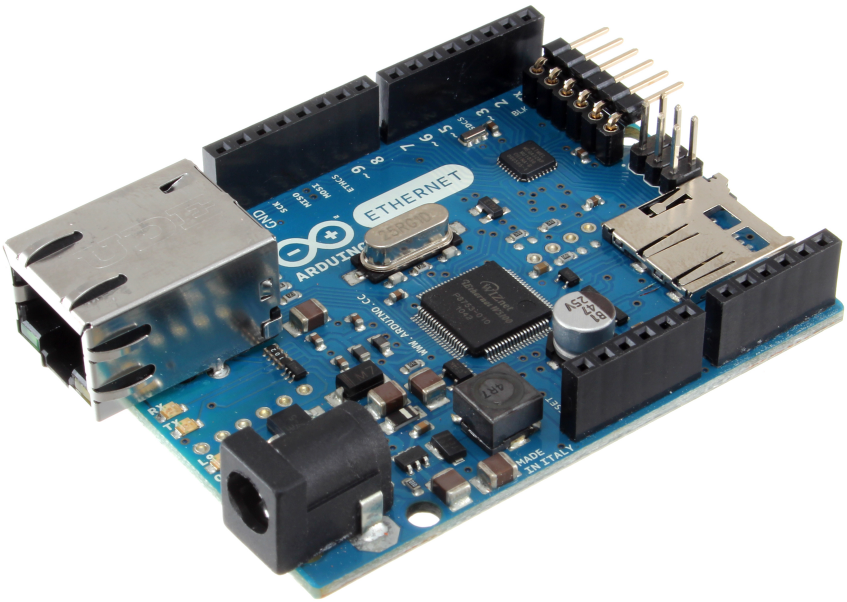
Gymnase de Burier

La lumière s'éteint-elle dans le frigo ?

Étude d'un bus en vue d'acquisition de données



Blonay, le 13 novembre 2012

Nathanaël Restori, 2M6

Remerciements

Je tiens à remercier M. Salanon pour ses conseils, son aide et pour m'avoir suivi tout au long de ce TM et M. Gelsomino pour m'avoir permis de le réaliser.

Je remercie aussi mes parents et ma sœur pour leur aide et leur relecture.

Table des matières

1. Introduction

1.1. Motivations

J'ai décidé de faire mon **Travail de Maturité (TM)**[1] sur ce sujet car je suis passionné de robotique depuis tout petit. J'ai commencé en participant à un atelier de soudure à la Maison Picson[2]. Je suis aussi allé à la première édition du Festival de Robotique à l'EPFL en 2008 et j'ai participé à deux ateliers (introduction à la programmation d'un microcontrôleur et soudure d'un robot DIDEL).

Je suis aussi passionné par l'informatique en général (autant au niveau matériel que logiciel), la programmation ainsi que les logiciels libres.

1.2. Présentation du travail

Pour mon travail, j'ai choisi un thème en rapport avec la robotique car ce domaine m'intéresse.

J'ai d'abord dû trouver un sujet plus précis. Suite à une discussion avec M. Salanon, je me suis tourné vers l'étude d'un **bus** de données (l'I²C) et la programmation d'une sorte de station météo.

J'ai dû choisir le matériel puis le commander aux États-Unis (le choix de matériel disponible en Suisse étant plus limité et pas forcément moins cher). J'ai ensuite étudié le fonctionnement du **bus** en théorie, puis « assemblé » et programmé la station. J'ai également réalisé un programme pour tester le fonctionnement de l'I²C à l'aide d'un oscilloscope.

J'ai fini par réaliser des mesures à l'aide de la station météo dans mon frigo et par vérifier le fonctionnement du **bus** en pratique.

1.3. Les principes du logiciel libre

Un logiciel libre est un logiciel qui respecte quatre libertés essentielles[3] :

1. la liberté d'exécuter le programme, pour tous les usages (liberté 0) ;
2. la liberté d'étudier le fonctionnement du programme, et de le modifier pour qu'il effectue vos tâches informatiques comme vous le souhaitez (liberté 1) ; l'accès au **code source** est une condition nécessaire ;

[1] Les mots en **gras** sont des mots définis dans le glossaire ou sont des acronymes.
[2] Établissement servant de cantine proposant aussi des activités à Blonay.
[3] FSF. *Qu'est-ce que le logiciel libre ?* URL : `https://www.gnu.org/philosophy/free-sw.fr.html`.

3. la liberté de redistribuer des copies, donc d'aider votre voisin (liberté 2) ;

4. la liberté de distribuer aux autres des copies de vos versions modifiées (liberté 3) ; en faisant cela, vous donnez à toute la communauté une possibilité de profiter de vos changements ; l'accès au **code source** est une condition nécessaire.

On retrouve aussi le terme « *open source* ». Le terme logiciel libre est défini par la **Free Software Foundation (FSF)**, tandis que celui d'open source est défini par l'**Open Source Initiative (OSI)**. En pratique, ces deux termes désignent la même chose.

Les sources des logiciels sont soumises à des licences spécifiques garantissant ces libertés. Les plus connues sont la **GNU General Public License (GPL)**, la **Berkeley Software Distribution (BSD)** ou la MIT (une copie de cette dernière est disponible à l'annexe A).

Quelques logiciels libres connus sont Firefox, LibreOffice/OpenOffice.org, Linux, Thunderbird et VLC.

Ce **TM** est donc entièrement articulé autour de logiciels et matériels libres : LaTeX et vim pour l'écriture de ce rapport, Arduino comme **plate-forme**, CMake et gcc pour la **compilation**, gnuplot pour les graphiques. Tout le code produit est donc placé sous licence MIT.

1.4. Conventions d'écriture

Les mots en **gras** sont des mots définis dans le glossaire ou sont des acronymes.

Les textes écrits avec `cette police d'écriture` sont des extraits de code.

2. L'I²C

Ce chapitre est basé sur un article de Wikipédia[1] ainsi qu'un autre article[2]. Toutes les images ont été faites par « Emiaille » et sont sous licence *Creative Commons paternité – partage à l'identique 3.0 (non transposée).*

2.1. Le bus I²C

Le **bus** I²C a été développé par Philips en 1982. La première norme (notée 1.0) a été publiée en 1992 et la dernière (notée 4.0) en 2012.

Quelques exemples d'utilisations courantes[3] :

- lire des données dans de la mémoire **Random Access Memory (RAM)** ;
- contrôler des convertisseurs de courant direct à continu et vice-versa ;
- changer le contraste, la teinte ou la balance des couleurs d'un écran ;
- changer le volume de haut-parleurs intelligents ;
- contrôler des écrans **Organic Light-Emitting Diode (OLED)** ou **Liquid Crystal Display (LCD)** ;
- lire des capteurs ;
- lire une **horloge temps réel** ;
- allumer ou éteindre une alimentation.

2.2. Les caractéristiques

L'I²C a plusieurs caractéristiques intéressantes :

- Il n'utilise que deux lignes (en réalité trois, la masse devant être commune à tous les maîtres et les esclaves).
- Il est multi-maîtres : plusieurs objets peuvent contrôler le **bus**.
- Il est multi-esclaves : plusieurs objets peuvent répondre à un maître.

[1] Wikipédia. *I²C*. URL : `https://fr.wikipedia.org/wiki/I2C`.
[2] Aurélien Jarno. *Le bus I²C*. URL : `http://www.aurel32.net/elec/i2c.php`.
[3] Wikipédia. *I²C*. URL : `http://en.wikipedia.org/wiki/I²C`.

- C'est un **bus** série : chaque information est découpée en plusieurs morceaux de taille fixe.
- C'est un **bus** synchrone : il possède une horloge propre (imposée par le maître qui veut parler).
- C'est un **bus** bidirectionnel half-duplex : les informations peuvent circuler dans les deux sens mais dans un seul à la fois.
- Il peut communiquer à des vitesses allant de 100 kbit/s à 5 Mbit/s.

2.3. Topologie

Tout d'abord, plusieurs équipements peuvent être connectés au **bus** en même temps (qu'ils soient maîtres ou esclaves).

Ces équipements sont connectés entre eux à l'aide de deux lignes :

- SDA (Serial Data Line) : ligne de données bidirectionnelle,
- SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Il faut toutefois une troisième ligne pour avoir une masse commune à tous les équipements. Ceux-ci doivent être alimentés avec le même potentiel (pour avoir la même référence comme niveau haut) mais peuvent être alimentés par différentes sources.

Les échanges ont toujours lieu entre un maître et un esclave et sont toujours débutés par le maître. Cependant, rien n'empêche à un équipement de passer du statut de maître au statut d'esclave et vice-versa.

Pour permettre à plusieurs maîtres de pouvoir imposer un niveau haut ou un niveau bas sur les lignes en même temps, des sorties à collecteur ouvert (ou à drain ouvert pour des circuits CMOS) sont utilisées. Deux résistances « pull-up » (R_P) tirent les lignes au niveau haut (V_{DD}).

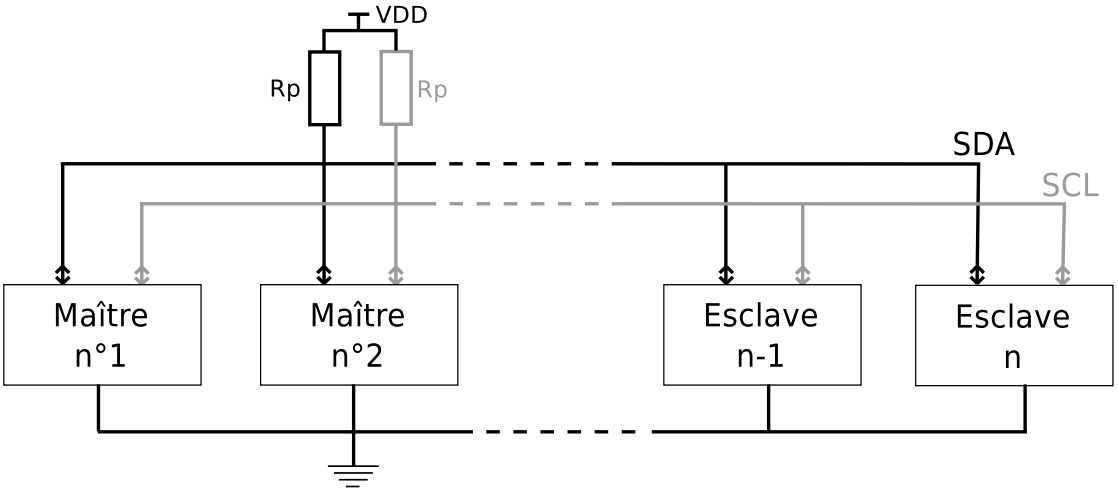


FIG. 2.1. : Architecture I²C avec plusieurs maîtres et plusieurs esclaves

2.3.1. Niveaux électriques

Les états logiques sont définis proportionnellement à la tension de la ligne et de la masse (tableau 2.1).

L'état logique « 0 » ou « LOW » est l'état « dominant », tandis que l'état logique « 1 » ou « HIGH » est l'état « récessif ». Cela veut dire que, si un équipement impose l'état « LOW » et qu'un autre impose l'état « HIGH », la ligne sera à l'état « LOW ».

État	Niveau
« LOW » ou « 0 »	-0.5 V à $0.3 \cdot V_{DD}$
« HIGH » ou « 1 »	$0.7 \cdot V_{DD}$ à V_{DD}

TAB. 2.1. : États en fonction du niveau de la ligne

2.4. Le protocole I²C

2.4.1. L'encodage

Tout d'abord, en informatique, les informations sont envoyées sous forme de « 0 » et de « 1 ». Un « 0 » ou un « 1 » s'appelle un bit. Souvent, les informations sont envoyées sous forme de paquets de huit bits. On appelle ces huit bits un octet (ou *byte* en anglais).

Pour transmettre un bit, le maître doit d'abord placer la ligne SCL au niveau « LOW » puis placer la ligne SDA au niveau voulu (« LOW » pour transmettre un « 0 » ou « HIGH » pour transmettre un « 1 »). Ensuite, il place la ligne SCL au niveau « HIGH », attend un temps défini par la vitesse utilisée (voir la sous-section suivante) puis replace la ligne SCL au niveau « LOW ». Un bit vient d'être transmis. Il recommence pour

transmettre le bit suivant. Tant que la ligne SCL est au niveau « HIGH », la ligne SDA ne doit pas changer d'état.

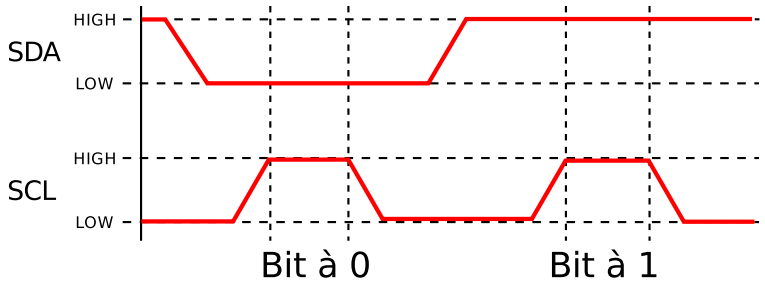


FIG. 2.2. : Encodage d'un bit I²C

2.4.2. Temps de pause et vitesse

Il existe cinq vitesses de transmission :

- « Standard mode » $\leq$ 100 kbit/s ;
- « Fast mode » $\leq$ 400 kbit/s ;
- « Fast plus mode » $\leq$ 1 Mbit/s ;
- « High-speed mode » $\leq$ 3,4 Mbit/s ;
- « Ultra-fast mode » $\leq$ 5 Mbit/s.

Pour chacune de ces vitesses, un temps de pause minimum est défini pour le niveau « LOW » et le niveau « HIGH » (voir le tableau 2.2). Pour les deux vitesses les plus élevées, il n'est pas défini de manière statique.

Mode	t_{LOWmin}	$t_{HIGHmin}$
Standard	4,7 s	4 s
Fast	1,3 s	0,6 s
Fast plus	0,5 s	0,26 s

TAB. 2.2. : Temps de pause minimum

2.4.3. La commande START

La commande START est une transgression à la règle d'encodage. Elle est utilisée pour signaler le début d'une **trame**. Pour envoyer un START, la ligne SDA doit passer de l'état « HIGH » à « LOW » pendant que la ligne SCL est à l'état « HIGH ».

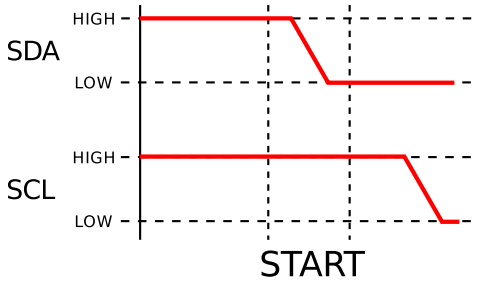


FIG. 2.3. : Condition de START I²C

2.4.4. La commande STOP

La commande STOP est une transgression à la règle d'encodage. Elle est utilisée pour signaler la fin d'une **trame**. Pour envoyer un STOP, la ligne SDA doit passer de l'état « LOW » à « HIGH » pendant que la ligne SCL est à l'état « HIGH ».

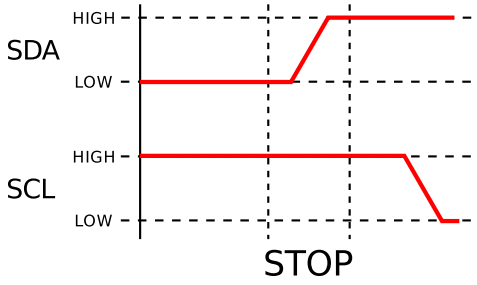


FIG. 2.4. : Condition de STOP I²C

2.4.5. La commande RESTART

La commande RESTART est une transgression à la règle d'encodage. Elle est utilisée pour signaler le début d'une nouvelle **trame** sans passer par une condition STOP. Pour envoyer un RESTART, la ligne SDA doit passer de l'état « HIGH » à « LOW » pendant que la ligne SCL est à l'état « HIGH ».

En fait, il s'agit de la commande START qui est envoyée entre un START et un STOP.

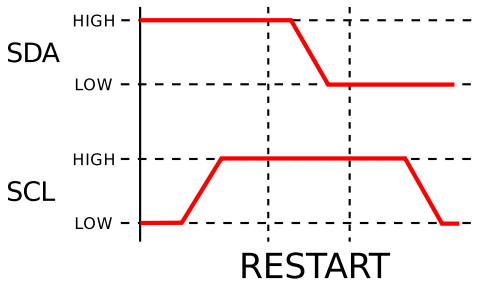


FIG. 2.5. : Condition de RESTART I²C

2.4.6. L'acquittement

Quand le récepteur a reçu un octet, il envoie la commande ACK pour signaler qu'il l'a bien reçu, ou la commande NACK pour signaler un problème lors de la réception. Quand le récepteur est un maître, il peut envoyer un NACK pour demander l'arrêt de la lecture avant d'envoyer un STOP. Pour envoyer un ACK, le récepteur envoie simplement un bit « 0 ». Pour envoyer un NACK, le récepteur envoie simplement un bit « 1 ».

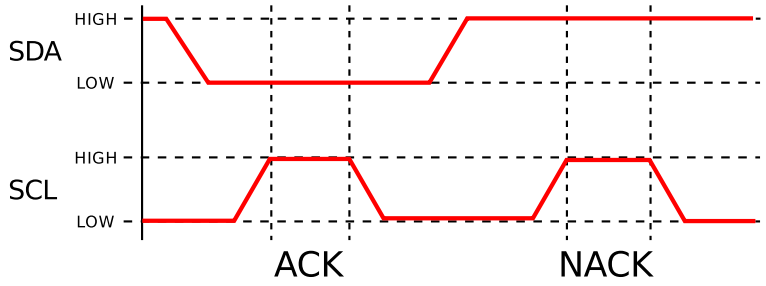


Fig. 2.6. : Acquittement I²C

2.4.7. La pause

À tout moment, l'esclave peut bloquer la ligne SCL à « LOW » pour signaler qu'il est occupé. Pour faire une pause, l'esclave maintient la ligne SCL au niveau « LOW » tandis que le maître maintient la ligne au niveau « HIGH ». Le maître va détecter l'écrasement et maintenir la ligne au niveau « HIGH » jusqu'à ce que l'esclave ait libéré la ligne.

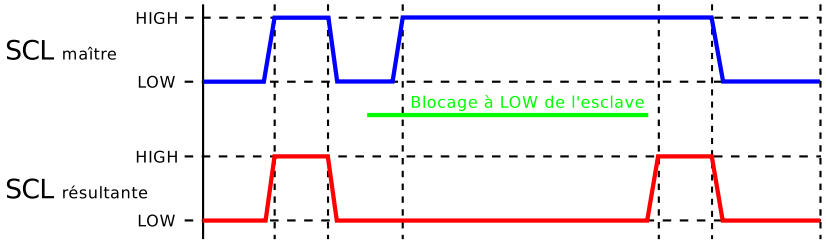


Fig. 2.7. : Pause I²C

2.4.8. L'adresse

Pour savoir à qui un maître veut parler, chaque esclave possède une adresse unique. Celle-ci peut être encodée sur 7 ou 10 bits.

L'adressage sur 7 bits

L'octet est composé de deux parties :

1. Les sept premiers bits correspondent à l'adresse.

2. Le dernier bit est appelé bit R/W (Read/Write) :
    - Si le maître envoie un « 1 », il demande une lecture et l'esclave lui envoie des données.
    - Si le maître envoie un « 0 », il demande une écriture et il envoie des données à l'esclave.

Il y a quelques adresses « spéciales »[4] :

- 00000000 : utilisée pour parler à tous les esclaves (appelée adresse de *broadcast* en anglais) ;
- 0000001X : utilisée pour accéder aux composants CBUS (ancêtre de l'I²C) ;
- 0000010X : réservée pour d'autres systèmes de **bus** ;
- 0000011X : réservée pour des utilisations futures ;
- 00001XXX : utilisée pour les composants haute-vitesse ;
- 11111XXX : réservée pour des utilisations futures ;
- 11110yzX : utilisée pour l'adressage sur 10 bits.

L'adressage sur 10 bits

Cette fois-ci, deux octets sont nécessaires :

- Le premier est l'octet « 11110yz0 ». Les bits y et z sont les bits de poids fort de l'adresse (autrement dit, ceux le plus à gauche). Le bit R/W est toujours à « 0 ».
- Le deuxième octet contient la suite de l'adresse sur huit bits. Il n'y a donc pas de bit R/W.

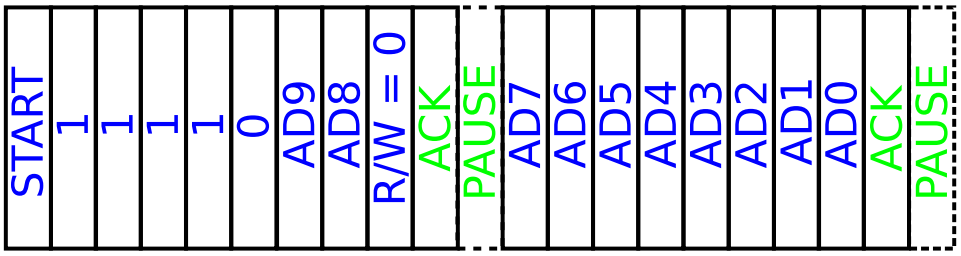


FIG. 2.8. : Adressage d'un esclave I²C sur 10 bits en écriture

Plusieurs esclaves peuvent avoir une adresse qui commence par yz. Si c'est le cas, ils répondent tous par un ACK en même temps. Une fois la suite de l'adresse envoyée, un

[4] Les X tout à droite correspondent au bit R/W. Les autres X, l'y et le z peuvent être soit un « 1 », soit un « 0 ».

seul esclave répondra par un ACK, l'adresse étant unique. Si le maître veut demander une lecture, il doit envoyer un RESTART à la fin du deuxième octet de l'adresse, puis envoyer l'octet « 11110yz1 ». Cette fois-ci, le bit R/W est à « 1 » et l'esclave saura que le maître demande une lecture.

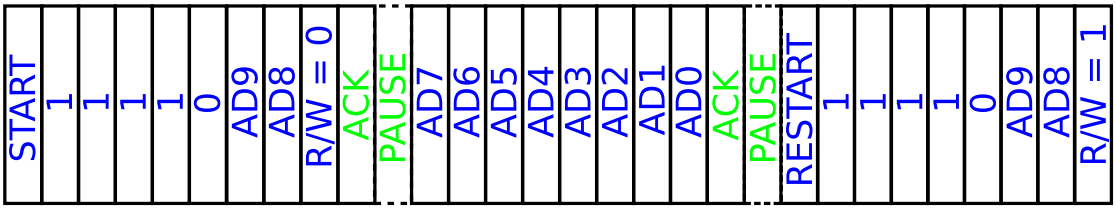


Fig. 2.9. : Adressage d'un esclave I²C sur 10 bits en lecture

2.4.9. Un échange complet

Tout d'abord, tous les maîtres écoutent en permanence les deux lignes. S'ils détectent un START, ils savent qu'ils doivent attendre un STOP avant de tenter de parler à un esclave. Cette écoute permanente permet aussi de détecter les pauses et les conflits (plusieurs maîtres qui tentent de parler en même temps).

Voici un exemple d'échange complet :

1. Le maître qui veut parler attend que le **bus** soit libre si ce dernier est occupé.
2. Le maître envoie la commande START.
3. Le maître envoie un octet : les sept premiers bits correspondent à l'adresse et le huitième permet de savoir si le maître demande une lecture ou une écriture (ici, une écriture).
4. L'esclave envoie un bit d'acquittement (ici, un ACK).
5. L'esclave peut demander une pause.
6. Le maître envoie un octet qui contient une commande.
7. L'esclave envoie un bit d'acquittement (ici, un ACK).
8. L'esclave peut demander une pause.
9. Le maître envoie la commande RESTART.
10. Le maître envoie un octet : les sept premiers bits correspondent à l'adresse et le huitième permet de savoir si le maître demande une lecture ou une écriture (ici, une lecture).
11. L'esclave envoie un premier octet qui contient le début des données.
12. Le maître envoie un bit d'acquittement (ici, un ACK).

13. L'esclave peut demander une pause.
14. L'esclave envoie un deuxième octet qui contient la suite des données.
15. Le maître envoie un bit d'acquittement (ici, un NACK).
16. L'esclave peut demander une pause.
17. Le maître envoie la commande STOP pour libérer le **bus**.

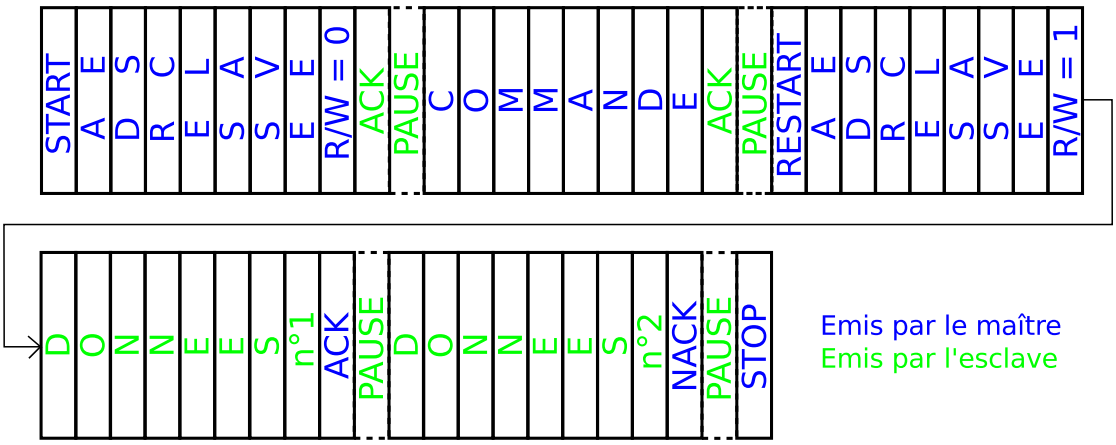


FIG. 2.10. : Exemple d'échange I²C entre un maître et un esclave

2.4.10. Le cas de conflit

Si deux maîtres (ou plus) prennent le contrôle du **bus** simultanément ou presque, les deux START et les lignes SCL vont se superposer et aucun des deux maîtres ne va se rendre compte qu'un autre est en train de parler en même temps. Toutefois, ils écoutent tous les deux pendant qu'ils écrivent. Tant qu'ils envoient un bit « 1 » ou « 0 » en même temps, il n'y aura pas de conflit entre les deux. Par contre, si l'un envoie un « 1 » et l'autre un « 0 », le « 0 » va écraser le « 1 » et le maître envoyant le « 1 » va détecter le conflit. Il va donc arrêter de parler et laisser l'autre continuer. Le conflit peut être détecté lors de l'écriture de l'adresse, du bit R/W ou lors de l'envoi d'une commande. Si les deux maîtres ont envoyé exactement la même chose, il n'y aura pas de conflit et ils liront ou écriront la même chose.

3. Le matériel

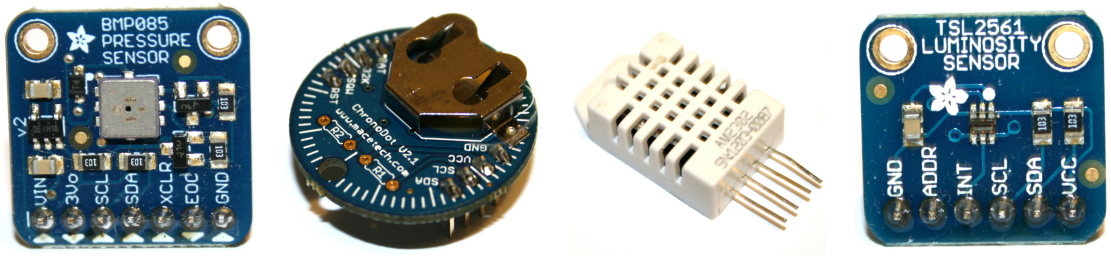
3.1. Le choix de la plate-forme

Il existe de nombreuses **plates-formes** en robotique : par exemple, le Boe-Bot de Parallax, utilisé dans les cours facultatifs de robotique de M. Gardon. J'ai fait le choix d'un Arduino car c'est une **plate-forme** de plus en plus répandue, peu chère (20 € pour une carte programmable). On trouve de nombreux exemples de **Do It Yourself (DIY)**, elle est programmable en **C++** (il est donc possible d'utiliser des **bibliothèques**) et c'est du **matériel libre**.

Pour pouvoir afficher les mesures des capteurs en direct et à travers le réseau, j'ai choisi un Arduino Ethernet (voir la photo de couverture).

3.2. Les capteurs

J'ai décidé de faire une station météo. Nous avons besoin d'un thermomètre (figure 3.1a), d'un baromètre (figure 3.1a aussi), d'un hygromètre (figure 3.1c) et d'un capteur de luminosité (figure 3.1d). J'ai ajouté une horloge (figure 3.1b) qui nous permettra d'avoir l'heure à laquelle les mesures ont été prises. Tous les capteurs ont été commandés sur le site d'Adafruit. L'hygromètre ne se connecte malheureusement pas en I²C, aucun capteur de ce type n'étant disponible.



(a) BMP085 (b) ChronoDot (c) DHT22 (d) TSL2561

FIG. 3.1. : Les capteurs

Photos personnelles

3.3. Les limitations

J'ai vite été confronté à une limitation : la mémoire de l'Arduino ne peut contenir que 32'256 octets de code compilé. Il ne peut donc pas contenir en même temps les **bibliothèques** pour utiliser les capteurs en I²C, le port RJ45 et la carte SD. La solution a été de couper le programme en deux : il y a donc un code pour afficher les mesures en direct à travers un navigateur web et un code pour prendre des mesures de manière régulière et les enregistrer.

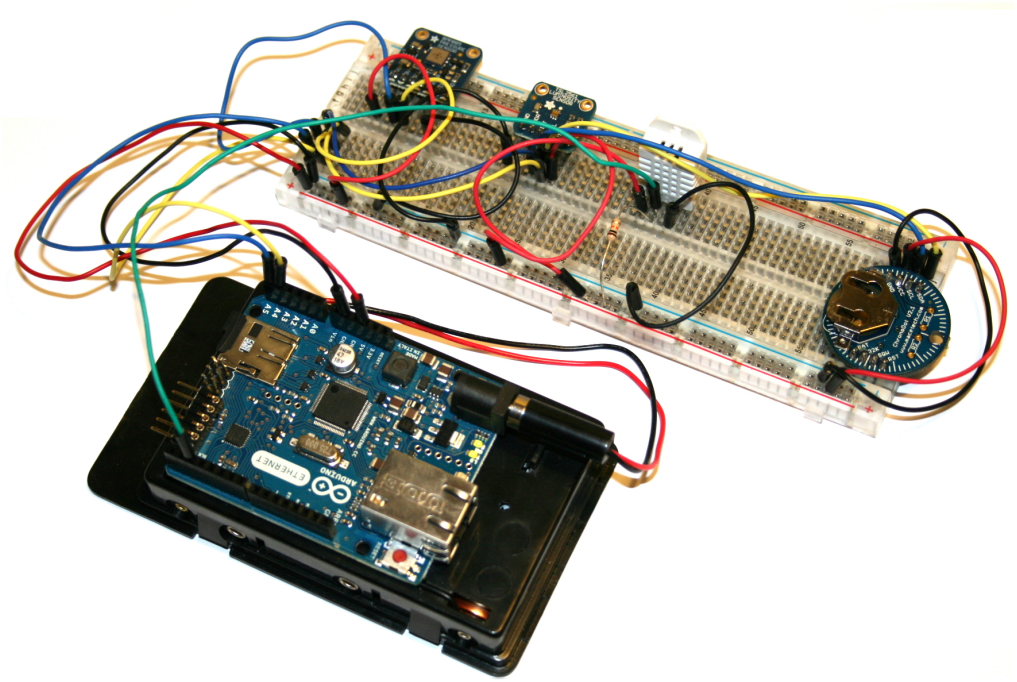


FIG. 3.2. : Montage complet

Photo personnelle

4. Le logiciel

4.1. En général

Les programmes sont composés de deux fonctions principales. La fonction `setup()` qui s'exécute une fois au démarrage du programme et la fonction `loop()` qui est exécutée en boucle. La première sert à initialiser et configurer les capteurs et les puces, tandis que la deuxième sert à traiter les informations.

Plusieurs **bibliothèques** sont utilisées. Comme bibliothèques internes[1], sont utilisées : `Ethernet` pour la puce de contrôle du port RJ45, `SD` ainsi que `SPI` pour la carte SD et `Wire` pour l'I²C. Comme bibliothèques tierces[2], sont utilisées : `BMP085` pour le capteur de température et de pression, `Chronodot` pour l'horloge, `DHT` pour le capteur d'humidité, `TSL2561` pour le capteur de luminosité et, pour finir, `WebServer` pour servir les pages web.

La partie logicielle est coupée en deux programmes : un enregistreur et un serveur web. Les sources sont disponibles respectivement à l'annexe B et à l'annexe D.

Un troisième programme sera utilisé pour tester en pratique le protocole I²C à l'aide d'un oscilloscope et pour vérifier si la théorie correspond à la pratique. Le **code source** est disponible à l'annexe H.

4.2. L'enregistreur

La fonction `setup()` commence par initialiser la carte SD, puis initialise les différents capteurs. Elle configure ensuite le capteur de luminosité et finit par vérifier si le fichier « DATA.TSV » existe. S'il n'existe pas, elle le crée et y enregistre les en-têtes (un exemple de ce fichier est disponible à l'annexe C).

La fonction `loop()` récupère les différentes données des capteurs puis les enregistre. Elle finit en faisant une pause d'un temps défini.

4.3. Le serveur web

La fonction `setup()` commence par initialiser la puce de contrôle du port RJ45 et le serveur web, configure ce dernier, puis initialise les différents capteurs. Elle configure ensuite le capteur de luminosité. Elle finit par appeler `sendNtpPacket()` qui va interroger un serveur **Network Time Protocol (NTP)** pour mettre à jour l'heure de l'horloge.

[1] Fournies avec l'Arduino.
[2] Fournies par d'autres personnes/enterprises/projets sur Internet.

La fonction `loop()` se contente de demander au serveur de s'occuper des connections entrantes. Ce dernier peut appeler deux fonctions : `defaultCmd()` si la page d'index est demandée ou `sensorsJsonCmd()` si c'est la page « sensors.json » qui est demandée.

La fonction `defaultCmd()` se contente d'envoyer la page d'index dont le **code source** est disponible à l'annexe E.

La fonction `sensorsJsonCmd()` récupère les valeurs des capteurs puis les envoie dans le format **JavaScript Object Notation (JSON)**. Un exemple est fourni à l'annexe F.

4.3.1. La page d'index

Une fois la page téléchargée par le navigateur web, du **JavaScript** est exécuté. Ce **JavaScript** va télécharger la page « sensors.json » de manière régulière, analyser son contenu puis mettre à jour le tableau qui contient les valeurs.

Une capture d'écran de la page est disponible à l'annexe G.

4.4. Tentative de réunion

J'ai tenté de réunir les deux programmes en un seul en allégeant les **bibliothèques** concernant la partie web. J'ai essayé de supprimer toutes les références au **Dynamic Host Configuration Protocol (DHCP)** et au **Domain Name System (DNS)**. Cela a permis de faire passer le programme final en dessous des 32'256 octets, mais il est apparu de drôles de choses : le programme gelait au bout de quelques secondes et la fonction `setup()` était appelée en boucle.

J'ai aussi cherché s'il était possible de mettre un système d'exploitation sur l'Arduino pour qu'il puisse charger les programmes depuis la carte SD, mais je n'ai rien trouvé.

4.5. Le programme pour tester en pratique l'I²C

La fonction `setup()` se contente d'initialiser la classe `Wire` afin d'utiliser le **bus**.

La fonction `loop()` va envoyer en boucle le nombre 0, 170 ou 255 à un esclave (ici, l'horloge) en écriture. L'opération est effectuée en boucle car un seul envoi est trop rapide pour être lu sur l'oscilloscope.

5. Test pratique de l'I²C

5.1. Le choix des nombres envoyés

Les trois nombres envoyés ont été choisis à cause de leur écriture en binaire[1]. Ce sont des cas « extrêmes » qui permettent de bien voir certains éléments sur les graphes de l'oscilloscope.

Notation décimale	Notation binaire
0	00000000
170	10101010
255	11111111

Tab. 5.1. : Équivalence décimal-binaire

5.2. Analyse des graphes

Pour pouvoir analyser les graphes plus facilement, j'ai rajouté le numéro ou le nom des bits sur la ligne d'horloge et le bit envoyé sur la ligne de données ainsi que le START et le STOP.

Les originaux sont disponibles à l'annexe I.

[1] Voir tableau 5.1.

5.2.1. Avec 0

La figure 5.1 montre l'écriture d'un 0 vue à l'oscilloscope. Le premier canal (en orange) montre la ligne des données (SDA) et le deuxième canal montre la ligne d'horloge (SCL).

On voit que la communication commence par l'envoi d'un START par le maître, suivi des sept bits de l'adresse (ici, « 1101000 ») puis du bit d'écriture (l'avant-dernier « 0 »). L'esclave envoie ensuite le bit d'acquittement (le dernier « 0 »). On remarque une crête, mais elle ne correspond pas à un RESTART. En effet : cette crête se situe alors que la ligne d'horloge est au niveau « LOW ». Après, on voit l'envoi du 0 (les huit premiers « 0 ») par le maître puis du bit d'acquittement par l'esclave. Le maître termine la communication par un STOP.

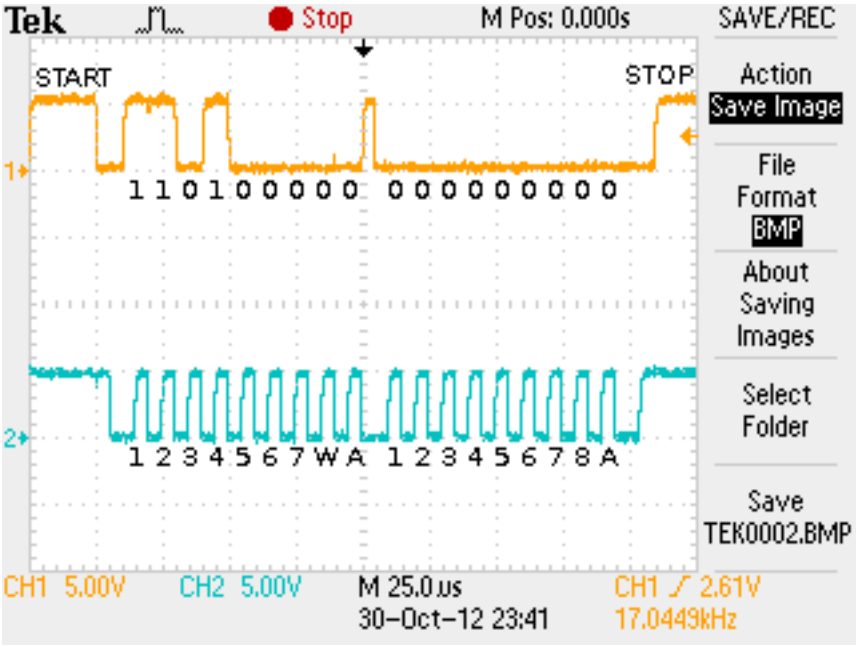


Fig. 5.1. : Écriture d'un 0 à l'oscilloscope

5.2.2. Avec 255

La figure 5.2 montre l'écriture d'un 255 vue à l'oscilloscope.

La communication commence de la même manière : envoi du START, de l'adresse, du bit d'écriture et pour finir, du bit d'acquittement. Pour le deuxième octet, on voit l'envoi du 255 (les huit premiers « 1 »), puis le bit d'acquittement. Mais cette fois-ci, ce dernier est positionné à « 1 ». L'esclave envoie donc un NACK (autrement dit, un non-acquittement). Le maître termine la communication avec un STOP. On remarque un creux entre le deuxième octet et le STOP. C'est pour pouvoir effectuer ce dernier. En effet : la ligne de données doit passer du niveau bas au niveau haut quand la ligne d'horloge est au niveau haut. La crête est toujours présente, mais « fusionnée » avec les « 1 » du deuxième octet.

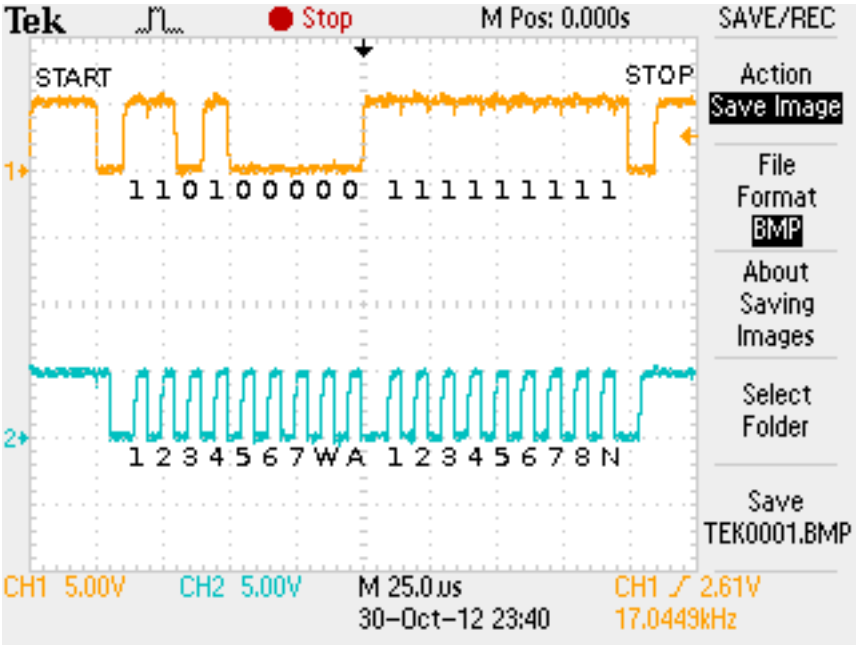


Fig. 5.2. : Écriture d'un 255 à l'oscilloscope

5.2.3. Avec 170

La figure 5.3 montre l'écriture d'un 170 vue à l'oscilloscope.

Encore une fois, la communication commence de la même manière. Pour le deuxième octet, on voit l'envoi du 170 (le « 10101010 »), puis le bit d'acquittement. Mais cette fois-ci encore, ce dernier est positionné à « 1 ». L'esclave envoie à nouveau un NACK. Le maître termine la communication avec un STOP.

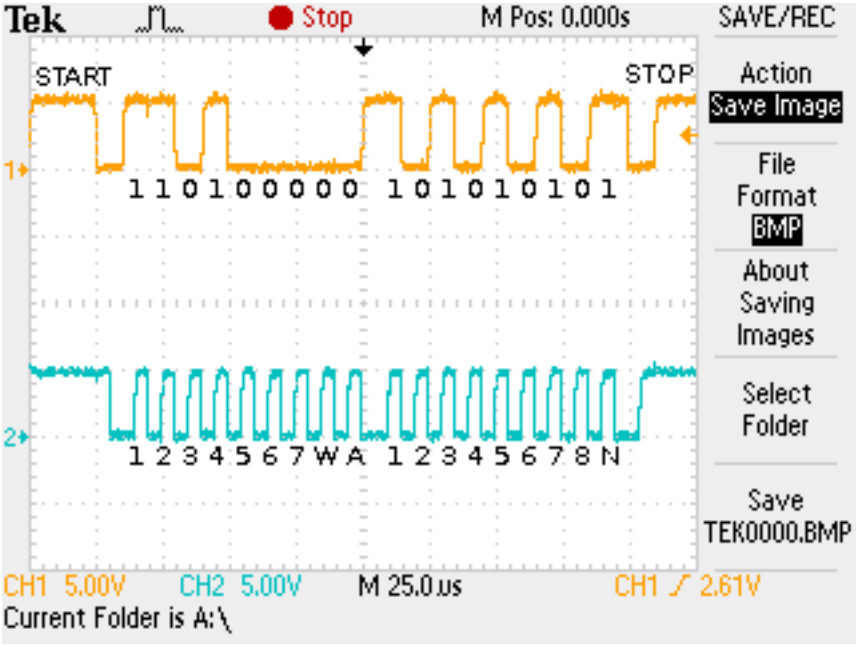


FIG. 5.3. : Écriture d'un 170 à l'oscilloscope

5.2.4. Explication du NACK

Pour expliquer les deux derniers NACK, il faut s'intéresser au fonctionnement de l'horloge[2] (pas la ligne mais le périphérique).

L'horloge possède un registre contenant les valeurs qui correspondent à l'heure, la date, la température et à deux alarmes. Ici, deux cas sont possibles : la lecture et l'écriture.

Si on veut lire une partie du registre, il faut commencer par envoyer en écriture la case à partir de laquelle on veut commencer à lire puis un STOP. Ensuite, on envoie une demande de lecture. L'horloge va répondre en envoyant un octet contenant les informations de la case sélectionnée. Ensuite, on peut soit envoyer un ACK pour recevoir la case suivante, soit envoyer un NACK pour arrêter la lecture.

Si l'on veut écrire, après avoir sélectionné la case, il faut continuer à écrire un octet. Cet octet sera écrit à l'endroit sélectionné. Deux choix s'offrent ensuite : écrire encore un octet qui sera placé dans la case suivante ou envoyer un STOP.

Les nombres envoyés pour mes essais sont donc des sélections de case. L'horloge a renvoyé un NACK car la case n° 170 ou 255 n'existe pas et la sélection a par conséquent échoué. Elle a, au contraire, renvoyé un ACK pour la case n° 0 car celle-ci existe. En effet : en informatique, les tableaux et les registres sont numérotés à partir de 0 et non de 1.

[2] Maxim Integrated. *DS3231*. URL : `http://datasheets.maximintegrated.com/en/ds/DS3231.pdf`.

6. Le cas du frigo

6.1. Les mesures

Pour accomplir cette expérience, la station météo a été placée dans un sachet plastique contenant du silica gel ainsi qu'une rallonge pour l'hygromètre pour qu'il puisse être exposé à l'humidité du frigo. La station météo a ensuite été placée pendant une heure au frigo. Voici mes premières constatations : premièrement, l'humidité ne dépassait jamais 70 %, deuxièmement, la température ne descendait jamais au dessous de 8 °C (voir la figure J.1).

Un deuxième essai a été réalisé, sans sachet plastique cette fois-ci. Le montage a été placé deux heures au frigo. Cette fois-ci, la température s'est stabilisée à environ 5 °C mais on constate une chute de la puissance après environ une heure. Cela est probablement dû au fait que les piles ne fournissent que peu d'énergie à basse température, la puissance augmentant une fois sortie du frigo (voir la figure J.2).

Un autre essai a ensuite été réalisé. Cette fois-ci, les piles ont été remplacées par un adaptateur qui fournissait du 9 volts. La station météo a été placée dans le frigo, pendant une nuit. Cette fois-ci, l'expérience s'est bien déroulée : il n'y a pas eu de perte de puissance.

6.2. Les résultats

Sur la première figure (6.1), on remarque plusieurs choses.

Tout d'abord, la température commence par descendre d'une manière qui semble exponentielle (ce n'est pas très visible ici mais l'est clairement sur un agrandissement) puis oscille de manière périodique entre 4 °C et 7.5 °C. On voit une légère augmentation de la température vers huit heures du matin.

La pression a diminué pendant la nuit, mais cela n'est pas lié au frigo.

L'humidité oscille aussi de manière périodique entre environ 40 % d'humidité et 70 % d'humidité après deux heures dans le frigo. On voit un grand pic vers huit heures du matin.

La luminosité est nulle tout au long de l'expérience sauf quand la station a été placée et retirée du frigo. On remarque un pic vers huit heures du matin. Cette augmentation et ces pics nous indiquent que la porte a été ouverte à cette heure-ci.

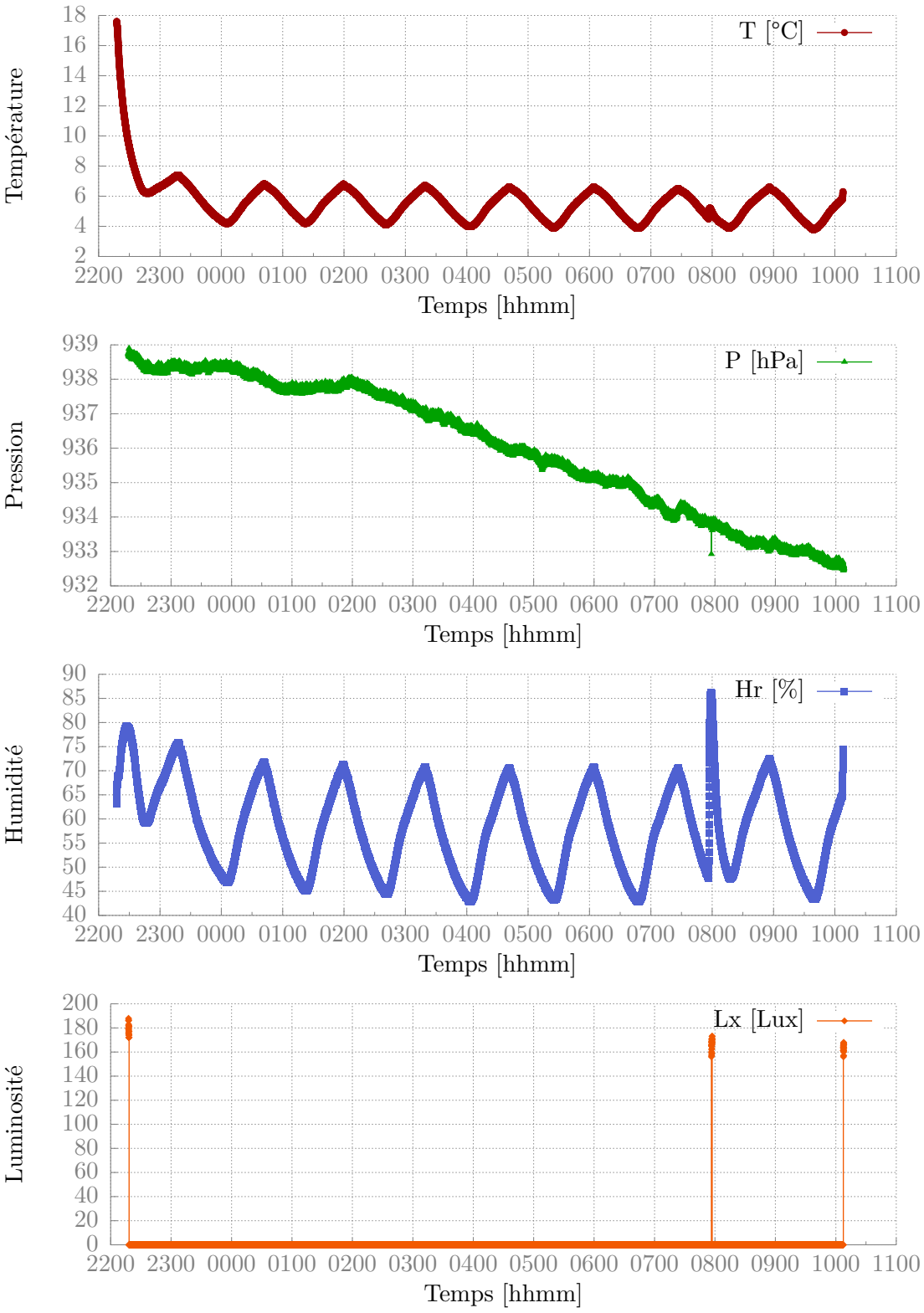


FIG. 6.1. : Les quatre graphiques

6.2.1. L'humidité en fonction de la température

La figure 6.2 représente l'humidité en fonction de la température entre trois heures et quart du matin et sept heures et demi du matin. D'après les cours de M. Bonnaz, je m'attendais à voir une courbe de pente négative, symétrique à celle-ci. En effet : plus l'air est chaud, plus il peut accueillir d'humidité, donc l'humidité relative devrait diminuer. Mais ça ne fonctionne que si l'on part du principe que l'eau ne se condense uniquement lorsque l'humidité atteint 100 % (ce qui était fait dans les exercices du cours).

Mais d'où peut donc venir cette différence ? Faisons un peu de calcul :

Par définition de l'humidité absolue[1] :

$$H = \frac{m}{V}$$

Or, $m = M \cdot n$ (cours de chimie). Nous avons donc :

$$H = \frac{n}{V} \cdot M_{eau}$$

Or, $\frac{n}{V} = \frac{P}{R \cdot T}$ (loi des gaz parfaits). Donc :

$$H_{éq} = \frac{P_{éq} \cdot M_{eau}}{R \cdot T}$$

Par définition de l'humidité relative :

$$H_r = \frac{H}{H_{éq}}$$

En remaniant les formules, on trouve :

$$m = V \cdot H = V \cdot H_r \cdot H_{éq} = V \cdot H_r \cdot \frac{P_{éq} \cdot M_{eau}}{R \cdot T}$$

Sachant que le frigo fait environ 200 litres et que :

- à 4 °C l'humidité relative est de 43 % et la pression à l'équilibre[2] est de 813 Pa ;
- à 6.6 °C l'humidité relative est de 70 % et la pression à l'équilibre est de 975 Pa ;

on peut calculer la quantité de vapeur dans le frigo à 4 et 6.6 °C.

À 4 °C, le frigo contient environ 0.55 gramme de vapeur et à 6.6 °C, il en contient 1.06 gramme. Si l'on part du principe que le frigo est étanche, la pente de la courbe peut donc s'expliquer par le fait que 0.51 gramme d'eau s'évapore et se condense à chaque cycle, ce qui est une quantité raisonnable.

[1] Une table des symboles est disponible à la page 34.
[2] Information que l'on trouve dans des tables.

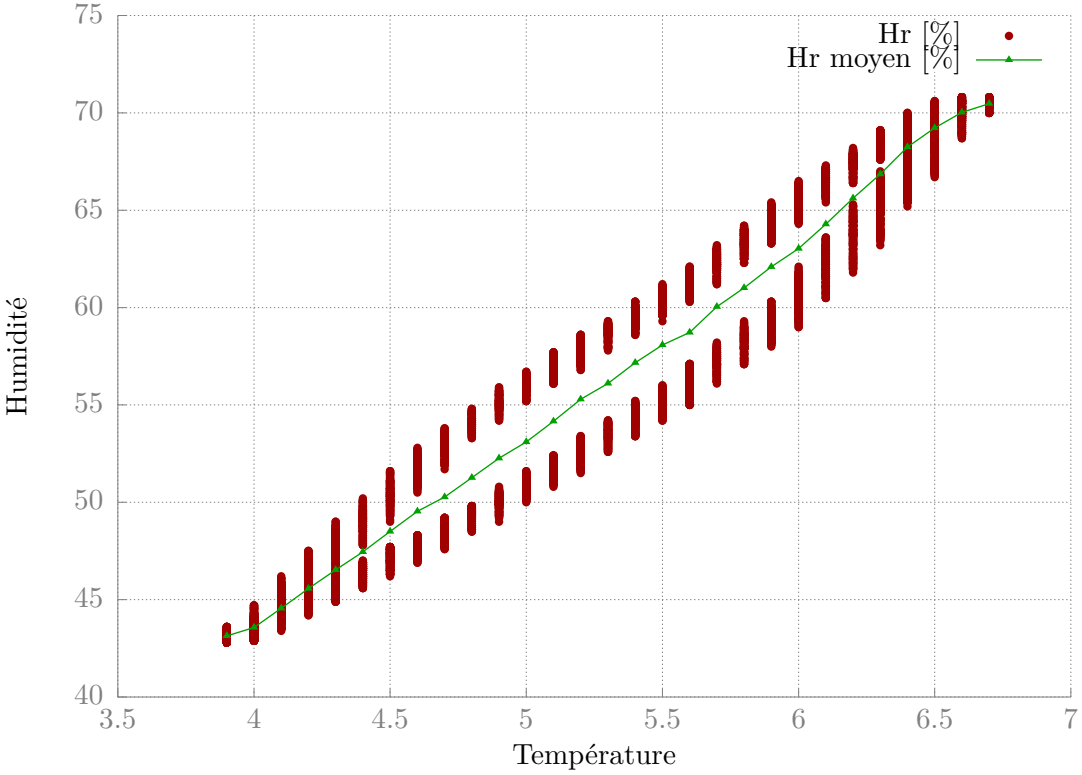


FIG. 6.2. : Humidité en fonction de la température

Table des figures

Liste des tableaux

Bibliographie

Sites web

FSF. *Qu'est-ce que le logiciel libre ?* URL : https://www.gnu.org/philosophy/free-sw.fr.html.
Jarno, Aurélien. *Le bus I²C*. URL : http://www.aurel32.net/elec/i2c.php.
Maxim Integrated. *DS3231*. URL : http://datasheets.maximintegrated.com/en/ds/DS3231.pdf.
Wikipédia. *I²C*. URL : https://fr.wikipedia.org/wiki/I2C.
– *I²C*. URL : http://en.wikipedia.org/wiki/I²C.

Images

Emiaille. *Acquittement I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_ACK.svg.
– *Adressage d'un esclave I²C sur 10 bits en écriture*. URL : https://commons.wikimedia.org/wiki/File:I2C_Adresse10bitsEcriture.svg.
– *Adressage d'un esclave I²C sur 10 bits en lecture*. URL : https://commons.wikimedia.org/wiki/File:I2C_Adresse10bitsLecture.svg.
– *Architecture I²C avec plusieurs maîtres et plusieurs esclaves*. URL : https://commons.wikimedia.org/wiki/File:I2C_Architecture.svg.
– *Condition de RESTART I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_RESTART.svg.
– *Condition de START I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_START.svg.
– *Condition de STOP I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_STOP.svg.
– *Encodage d'un bit I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_Encodage.svg.
– *Exemple d'échange I²C entre un maître et un esclave*. URL : https://commons.wikimedia.org/wiki/File:I2C_EchangeMaitreEsclave.svg.
– *Pause I²C*. URL : https://commons.wikimedia.org/wiki/File:I2C_Pause.svg.
oomlout. *Arduino Ethernet*. URL : https://commons.wikimedia.org/wiki/File:Arduino_Ethernet_Board_.jpg.

Acronymes

BSD
Berkeley Software Distribution. 5

DHCP
Dynamic Host Configuration Protocol. 18, *Glossaire :* DHCP

DIY
Do It Yourself. 15, *Glossaire :* DIY

DNS
Domain Name System. 18, *Glossaire :* DNS

FSF
Free Software Foundation. 5

GPL
GNU General Public License. 5

JSON
JavaScript Object Notation. 18

LCD
Liquid Crystal Display. 6

NTP
Network Time Protocol. 17

OLED
Organic Light-Emitting Diode. 6

OSI
Open Source Initiative. 5

RAM
Random Access Memory. 6

TM
Travail de Maturité. 4, 5

Glossaire

bibliothèque
ensemble de fonctions permettant de manipuler un capteur, un objet, une puce (etc.) plus facilement. 15–18

bus
un bus (en informatique) est un système de communication entre les composants d'un ordinateur. Il désigne autant le support physique, le logiciel et le protocole[3]. 4, 6, 7, 12–14, 18

C++
langage de programmation ancien et répandu. 15

code source
on peux considérer le code source comme la « recette de cuisine » du programme. Il décrit ce que doit faire le programme, quand et comment. 4, 5, 17, 18

compilation
transformation du code source (lisible pour un humain) en code machine (lisible pour une machine). 5

DHCP
protocole servant à attribuer une adresse IP à un ordinateur quand il le demande. 18

DIY
littéralement, « faites-le vous-même ». S'assimile à du bricolage. 15

DNS
protocole servant à relier un nom de domaine (www.gymnasedeburier.ch) à une adresse IP (84.16.80.62). 18

horloge temps réel
horloge permettant un décompte très précis du temps, utilisée en électronique. 6

JavaScript
language permettant de créer des scripts exécutés par le navigateur web. 18

[3]Tiré de Wikipédia.

matériel libre
: matériel dont les plans sont publics. Tout le monde peut les modifier, fabriquer, redistribuer et utiliser. Équivalent matériel du logiciel libre. 15

plate-forme
: en informatique, une base de travail à partir de laquelle on peut écrire, lire, utiliser, développer un ensemble de logiciels[4]. 5, 15

trame
: une trame est un bloc d'information véhiculé au travers d'un support physique[5]. 9, 10

[4]Tiré de Wikipédia.
[5]Tiré de Wikipédia.

Symboles

H	Humidité [g/m³]	26
$H_{éq}$	Humidité à l'équilibre [g/m³]	26
H_r	Humidité relative	26
M	Masse molaire [g/mol]	26
m	Masse de vapeur [g]	26
n	Quantité de matière en moles [mol]	26
P	Pression de vapeur [Pa]	26
$P_{éq}$	Pression de vapeur à l'équilibre [Pa]	26
R	Constante universelle des gaz parfaits	26
T	Température de l'air [K]	26
V	Volume d'air [m³]	26

Annexes

A. Licence MIT

Copyright (C) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions :

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

B. WeatherStationLogger.ino

```
/*
   Copyright (C) 2012 Nathanaël Restori

   Permission is hereby granted, free of charge, to any person
   obtaining a copy of this software and associated documentation
   files (the "Software"), to deal in the Software without
   restriction, including without limitation the rights to use, copy,
   modify, merge, publish, distribute, sublicense, and/or sell copies
   of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions :

   The above copyright notice and this permission notice shall be
   included in all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
   EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
   SOFTWARE.

*/

#include <SD.h>
#include <Wire.h>
#include "BMP085.h"
#include "Chronodot.h"
#include "DHT.h"
#include "TSL2561.h"

#define TIMING  1000 // milliseconds
#define DHTPIN  2
#define DHTTYPE DHT22
#define SDPIN   4

```

```
38 BMP085    bmp;
39 DHT       dht(DHTPIN, DHTTYPE);
40 TSL2561   tsl(TSL2561_ADDR_FLOAT);
41 Chronodot chronodot;
42 File      file;
43
44 void setup() {
45     Serial.begin(9600);
46
47     //----- SD Card
48     Serial.println("INIT SD Card");
49
50     if (!SD.begin(4)) {
51         Serial.println("FAILED");
52         return;
53     }
54     Serial.println("DONE");
55
56     //----- Sensors
57     Serial.println("INIT Sensors");
58
59     bmp.begin();
60     chronodot.begin();
61     dht.begin();
62     tsl.begin();
63
64     //tsl.setGain(TSL2561_GAIN_0X); // bright situtations
65     tsl.setGain(TSL2561_GAIN_16X);  // dim situations
66
67     tsl.setTiming(TSL2561_INTEGRATIONTIME_13MS);    // bright light
68     //tsl.setTiming(TSL2561_INTEGRATIONTIME_101MS); // medium light
69     //tsl.setTiming(TSL2561_INTEGRATIONTIME_402MS); // dim light
70
71     if (!SD.exists("data.tsv")) {
72         file = SD.open("data.tsv", FILE_WRITE);
73
74         if (file) {
75             Serial.print("Creating data.tsv");
76             file.println("#Time\t\t\tT [°C]\tP [Pa]\tAlt [m]\tHr [%]\t"
77                     "Lx [lux]");
78             file.close();
79             Serial.println("DONE");
80         } else {
81             Serial.println("Error opening data.tsv");
```

```
82          }
83      }
84  }
85
86  void loop() {
87      float   temperature = bmp.readTemperature();
88      int32_t pressure    = bmp.readPressure();
89      float   altitude    = bmp.readAltitude();
90
91      float humidity = dht.readHumidity();
92
93      uint32_t lum = tsl.getFullLuminosity();
94      uint16_t lightIr   = lum >> 16;
95      uint16_t lightFull = lum & 0xFFFF;
96
97      DateTime time = chronodot.now();
98
99      file = SD.open("data.tsv", FILE_WRITE);
100
101     if (file) {
102         Serial.print("Writing to data.tsv...");
103         file.print(time.year());   file.print("-");
104         file.print(time.month());  file.print("-");
105         file.print(time.day());    file.print(" ");
106         file.print(time.hour());   file.print(" :");
107         file.print(time.minute()); file.print(" :");
108         file.print(time.second()); file.print("\t");
109
110         file.print(temperature); file.print("\t");
111         file.print(pressure);    file.print("\t");
112         file.print(altitude);    file.print("\t");
113         file.print(humidity);    file.print("\t");
114         file.println(tsl.calculateLux(lightFull, lightIr));
115         file.close();
116         Serial.println("DONE");
117     } else {
118         Serial.println("Error opening data.tsv");
119     }
120
121     delay(TIMING);
122 }
```

C. DATA.TSV

#Time	T [°C]	P [Pa]	Alt [m]	Hr [%]	Lx [lux]
2012-9-20 20 :6 :46	22.30	94791	559.22	38.00	34
2012-9-20 20 :6 :47	22.30	94785	559.05	38.00	35
2012-9-20 20 :6 :49	22.30	94785	559.13	38.00	35
2012-9-20 20 :6 :50	22.20	94789	558.34	38.00	36
2012-9-20 20 :6 :52	22.20	94789	558.87	38.60	35
2012-9-20 20 :6 :53	22.10	94791	559.05	38.60	36
2012-9-20 20 :6 :54	22.00	94787	559.05	38.50	37
2012-9-20 20 :6 :56	22.00	94784	559.05	38.50	37
2012-9-20 20 :6 :57	21.90	94784	559.31	38.60	37
2012-9-20 20 :6 :58	21.90	94790	559.05	38.60	36
2012-9-20 20 :7 :0	21.80	94785	559.31	38.60	36
2012-9-20 20 :7 :1	21.80	94787	559.13	38.60	38
2012-9-20 20 :7 :3	21.70	94789	558.61	39.00	37
2012-9-20 20 :7 :4	21.60	94786	558.96	39.00	38
2012-9-20 20 :7 :5	21.60	94778	559.57	39.00	0
2012-9-20 20 :7 :7	21.50	94777	560.63	39.00	0
2012-9-20 20 :7 :8	21.50	94775	560.01	39.00	0
2012-9-20 20 :7 :10	21.40	94786	559.57	39.00	0
2012-9-20 20 :7 :11	21.30	94780	559.22	38.90	0
2012-9-20 20 :7 :12	21.30	94790	559.40	38.90	0
2012-9-20 20 :7 :14	21.20	94783	559.22	38.70	0
2012-9-20 20 :7 :15	21.20	94787	558.87	38.70	0
2012-9-20 20 :7 :16	21.10	94780	559.92	38.60	0
2012-9-20 20 :7 :18	21.00	94785	559.40	38.60	0
2012-9-20 20 :7 :19	21.00	94788	559.66	38.40	0
2012-9-20 20 :7 :21	20.90	94782	559.49	38.40	0
2012-9-20 20 :7 :22	20.90	94790	559.31	38.40	0
2012-9-20 20 :7 :23	20.80	94790	558.87	38.40	0
2012-9-20 20 :7 :25	20.80	94789	559.05	38.30	0
2012-9-20 20 :7 :26	20.70	94790	559.22	38.30	0
2012-9-20 20 :7 :28	20.60	94786	559.05	38.10	0
2012-9-20 20 :7 :29	20.60	94779	559.49	38.10	0
2012-9-20 20 :7 :30	20.50	94779	559.49	38.00	0
2012-9-20 20 :7 :32	20.50	94781	559.57	38.00	0
2012-9-20 20 :7 :33	20.40	94781	559.49	38.00	0
2012-9-20 20 :7 :34	20.40	94783	559.22	38.00	0

D. WeatherStationWeb.ino

```
/*
   Copyright (C) 2012 Nathanaël Restori

   Permission is hereby granted, free of charge, to any person
   obtaining a copy of this software and associated documentation
   files (the "Software"), to deal in the Software without
   restriction, including without limitation the rights to use, copy,
   modify, merge, publish, distribute, sublicense, and/or sell copies
   of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions :

   The above copyright notice and this permission notice shall be
   included in all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
   EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
   SOFTWARE.

*/

#include <Wire.h>
#include <SPI.h>
#include <Ethernet.h>
#include "BMP085.h"
#include "Chronodot.h"
#include "DHT.h"
#include "TSL2561.h"
#include "WebServer.h"

#define TIMEZONE        7200 // 2 hours
#define DHTPIN          2
#define DHTTYPE         DHT22
```

```
38 #define PREFIX          ""
39 #define NTP_PACKET_SIZE 48
40
41 static uint8_t   mac[]      = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
42 static IPAddress ip         = IPAddress(192, 168, 1, 210);
43 static IPAddress timeserver = IPAddress(213, 239, 239, 164);
44
45 P(indexHtml) = "<!DOCTYPE html><html><head><meta charset='utf-8'/"
46                "><title>Station Météo</title><style>html, body{ba"
47                "ckground : #000;color : #fff;}body{font-family : 'De"
48                "jaVu Sans', Verdana, Arial, sans-serif;font-size :"
49                " 100%;margin : 0;padding : 0;}#home{background : #00"
50                "0;color : #fff;margin : 1em;padding : 0;}#home h1{co"
51                "lor : #f7c000;font-size : 2em;margin : 1em;text-alig"
52                "n : center;}#home p{background : #111;border : 1px s"
53                "olid #444;font-size : 1.5em;list-style-type : none;"
54                "padding : 0;width : 100%;text-align : center;}#home "
55                "table{background : #111;border : 1px solid #444;fon"
56                "t-size : 1.5em;list-style-type : none;padding : 0;wi"
57                "dth : 100%;text-align : center;}#footer{background :"
58                " #000;color : #888;font-size : 0.8em;margin : 1em;te"
59                "xt-align : center;padding : 0;}</style></head><body"
60                "><div id='home'><h1>Station Météo</h1><table><tr>"
61                "<th>Temps</th><th>T [°C]</th><th>P [Pa]</th><th>A"
62                "lt [m]</th><th>Hr [%]</th><th>Lx [lux]</th></tr><"
63                "tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0<"
64                "/td><td>0</td></tr></table><p><label>Intervale de"
65                " mise à jour (en secondes) : </label><input id='i"
66                "nterval' type='number' value='2'/></p></div><div "
67                "id='footer'> par Nathanaël Restori </div><script>"
68                "function updateInfo(){var xhr=new XMLHttpRequest("
69                ");xhr.open('GET', '/sensors.json');xhr.onreadysta"
70                "techange=function(){if (xhr.readyState==4 && xhr."
71                "status==200){var response=JSON.parse(xhr.response"
72                "Text);var rows=document.getElementsByTagName('tr'"
73                ");var cell=rows[1].firstChild;var index=0;while ("
74                "cell){cell.firstChild.nodeValue=response.sensors["
75                "index].data;cell=cell.nextSibling;index++;}};xhr"
76                ".send(null);}updateInfo();var updateID=setInterva"
77                "l(updateInfo, 2000);var interval=document.getElem"
78                "entById('interval');interval.addEventListener('ch"
79                "ange', function(e){clearInterval(updateID);update"
80                "ID=setInterval(updateInfo, e.target.value*1000);}"
81                ", true);</script></body></html>";
```

```
82
83  // no-cost stream operator as described at
84  // http ://sundial.org/arduino/?page_id=119
85      template<class T>
86  inline Print &operator <<(Print &obj, T arg)
87  { obj.print(arg); return obj; }
88
89  BMP085    bmp;
90  Chronodot chronodot;
91  DHT       dht(DHTPIN, DHTTYPE);
92  TSL2561   tsl(TSL2561_ADDR_FLOAT);
93  WebServer webserver(PREFIX, 80);
94
95  byte packetBuffer[ NTP_PACKET_SIZE];
96  EthernetUDP Udp;
97
98  void defaultCmd(WebServer &server, WebServer ::ConnectionType type,
99          char *url_tail, bool tail_complete) {
100     server.httpSuccess();
101     server.printP(indexHtml);
102 }
103
104 void sensorsJsonCmd(WebServer &server, WebServer ::ConnectionType type,
105         char *url_tail, bool tail_complete) {
106     float   temperature = bmp.readTemperature();
107     int32_t pressure    = bmp.readPressure();
108     float   altitude    = bmp.readAltitude();
109
110     float humidity = dht.readHumidity();
111
112     uint32_t lum = tsl.getFullLuminosity();
113     uint16_t lightIr   = lum >> 16;
114     uint16_t lightFull = lum & 0xFFFF;
115
116     DateTime time = chronodot.now();
117
118     server.httpSuccess("application/json");
119
120     server << "{ \"sensors\" : [" << "{\"data\" :\""
121         << time.hour()   << " :"
122         << time.minute() << " :"
123         << time.second() << " "
124         << time.day()    << "/"
125         << time.month()  << "/"
```

```
126         << time.year()   << "\"},"
127
128         << "{\"data\" :" << temperature << "},"
129         << "{\"data\" :" << pressure    << "},"
130         << "{\"data\" :" << altitude    << "},"
131         << "{\"data\" :" << humidity    << "},"
132         << "{\"data\" :" << tsl.calculateLux(lightFull, lightIr) << "}"
133         << "] }";
134 }
135
136 // Taken from http ://arduino.cc/en/Tutorial/UdpNtpClient .
137 void sendNTPpacket() {
138     // set all bytes in the buffer to 0
139     memset(packetBuffer, 0, NTP_PACKET_SIZE);
140     // Initialize values needed to form NTP request
141     // (see URL above for details on the packets)
142     packetBuffer[0] = 0b11100011;   // LI, Version, Mode
143     packetBuffer[1] = 0;     // Stratum, or type of clock
144     packetBuffer[2] = 6;     // Polling Interval
145     packetBuffer[3] = 0xEC;  // Peer Clock Precision
146     // 8 bytes of zero for Root Delay & Root Dispersion
147     packetBuffer[12]  = 49;
148     packetBuffer[13]  = 0x4E;
149     packetBuffer[14]  = 49;
150     packetBuffer[15]  = 52;
151
152     // all NTP fields have been given values, now
153     // you can send a packet requesting a timestamp :
154     Udp.beginPacket(timeserver, 123); //NTP requests are to port 123
155     Udp.write(packetBuffer,NTP_PACKET_SIZE);
156     Udp.endPacket();
157     Serial.println("Sending ntp packet");
158
159     delay(1000);
160     if ( Udp.parsePacket() ) {
161         // We've received a packet, read the data from it
162         Udp.read(packetBuffer,NTP_PACKET_SIZE);  // read the packet
163                                                  // into the buffer
164
165         // the timestamp starts at byte 40 of the received packet and is
166         // four bytes, or two words, long. First, esxtract the two
167         // words :
168
169         unsigned long highWord = word(packetBuffer[40],
```

```
170                 packetBuffer[41]);
171         unsigned long lowWord = word(packetBuffer[42],
172                 packetBuffer[43]);
173         // combine the four bytes (two words) into a long integer
174         // this is NTP time (seconds since Jan 1 1900) :
175         unsigned long secsSince1900 = highWord << 16 | lowWord;
176         // Unix time starts on Jan 1 1970. In seconds, that's
177         // 2208988800 :
178         const unsigned long seventyYears = 2208988800UL;
179         // subtract seventy years :
180         unsigned long epoch = secsSince1900 - seventyYears + TIMEZONE;
181
182         Serial.println("Adjusting time");
183         chronodot.adjust(DateTime(epoch));
184         return;
185     }
186     Serial.println("No reponse");
187 }
188
189 void setup() {
190     Serial.begin(9600);
191
192     //----- Ethernet/WebServer
193     Serial.println("INIT Ethernet");
194
195     Ethernet.begin(mac, ip);
196     webserver.begin();
197
198     webserver.setDefaultCommand(&defaultCmd);
199     webserver.addCommand("sensors.json", &sensorsJsonCmd);
200
201     //----- Sensors
202     Serial.println("INIT Sensors");
203
204     bmp.begin();
205     dht.begin();
206     tsl.begin();
207     chronodot.begin();
208
209     //tsl.setGain(TSL2561_GAIN_0X); // bright situtations
210     tsl.setGain(TSL2561_GAIN_16X);  // dim situations
211
212     tsl.setTiming(TSL2561_INTEGRATIONTIME_13MS);    // bright light
213     //tsl.setTiming(TSL2561_INTEGRATIONTIME_101MS); // medium light
```

```
214     //tsl.setTiming(TSL2561_INTEGRATIONTIME_402MS); // dim light
215
216     Udp.begin(8888);
217     sendNTPpacket();
218 }
219
220 void loop() {
221     webserver.processConnection();
222
223     delay(500);
224 }
```

E. index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8' />
    <title>Station Météo</title>
    <style>
      html, body {
        background : #000;
        color : #fff;
      }

      body {
        font-family : 'DejaVu Sans', Verdana, Arial, sans-serif;
        font-size : 100%;
        margin : 0;
        padding : 0;
      }

      #home {
        background : #000;
        color : #fff;
        margin : 1em;
        padding : 0;
      }

      #home h1 {
        color : #f7c000;
        font-size : 2em;
        margin : 1em;
        text-align : center;
      }

      #home p {
        background : #111;
        border : 1px solid #444;
        font-size : 1.5em;
        list-style-type : none;
```

```
38          padding : 0;
39          width : 100%;
40          text-align : center;
41        }
42
43        #home table {
44          background : #111;
45          border : 1px solid #444;
46          font-size : 1.5em;
47          list-style-type : none;
48          padding : 0;
49          width : 100%;
50          text-align : center;
51        }
52
53        #footer {
54          background : #000;
55          color : #888;
56          font-size : 0.8em;
57          margin : 1em;
58          text-align : center;
59          padding : 0;
60        }
61      </style>
62    </head>
63    <body>
64      <div id='home'>
65        <h1>Station Météo</h1>
66        <table>
67          <tr>
68            <th>Temps</th><th>T [°C]</th><th>P [Pa]</th><th>Alt
69              [m]</th><th>Hr [%]</th><th>Lx [lux]</th>
70          </tr>
71          <tr>
72            <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td>
73          </tr>
74        </table>
75        <p>
76        <label>Intervale de mise à jour (en secondes) : </label>
77        <input id='interval' type='number' value='2' />
78        </p>
79      </div>
80      <div id='footer'>
81        par Nathanaël Restori
```

```
82      </div>
83
84      <script>
85        function updateInfo() {
86          var xhr = new XMLHttpRequest();
87          xhr.open('GET', '/sensors.json');
88          xhr.onreadystatechange = function() {
89            if (xhr.readyState == 4 && xhr.status == 200) {
90              var response = JSON.parse(xhr.responseText);
91
92              var rows = document.getElementsByTagName('tr');
93              var cell = rows[1].firstChild;
94              var index = 0;
95              while (cell) {
96                cell.firstChild.nodeValue = response.sensors[index].data;
97                cell = cell.nextSibling;
98                index++;
99              }
100           }
101         };
102         xhr.send(null);
103       }
104
105       updateInfo();
106
107       var updateID = setInterval(updateInfo, 2000);
108
109       var interval = document.getElementById('interval');
110       interval.addEventListener('change', function(e) {
111         clearInterval(updateID);
112         updateID = setInterval(updateInfo, e.target.value*1000);
113       }, true);
114     </script>
115   </body>
116 </html>
```

F. sensors.json

```
{
  "sensors" :[
  {
    "data" :"17 :56 :0 13/9/2012"
  },
  {
    "data" :21.50
  },
  {
    "data" :94298
  },
  {
    "data" :602.11
  },
  {
    "data" :38.30
  },
  {
    "data" :30
  }
  ]
}
```

G. Capture d'écran de la page d'index

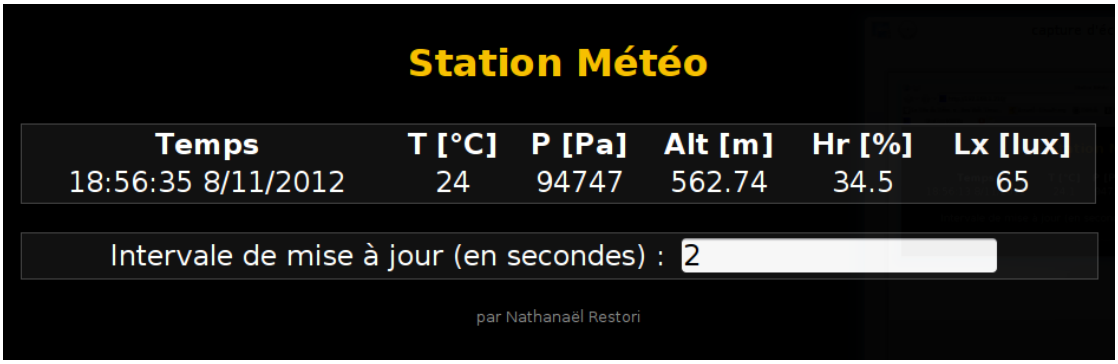


FIG. G.1. : Capture d'écran de la page d'index

H. I2CPractise.ino

```
/*
   Copyright (C) 2012 Nathanaël Restori

   Permission is hereby granted, free of charge, to any person
   obtaining a copy of this software and associated documentation
   files (the "Software"), to deal in the Software without
   restriction, including without limitation the rights to use, copy,
   modify, merge, publish, distribute, sublicense, and/or sell copies
   of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions :

   The above copyright notice and this permission notice shall be
   included in all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
   EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
   MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
   SOFTWARE.

*/

#include <Wire.h>

#define CHRONODOT_ADDRESS 0x68

void setup() {
    Wire.begin();
}

void loop() {
    Wire.beginTransmission(CHRONODOT_ADDRESS);
    Wire.write((byte)0);
    //Wire.write((byte)170);
```

```
38      //Wire.write((byte)255);
39      Wire.endTransmission();
40  }
```

I. Originaux de l'oscilloscope

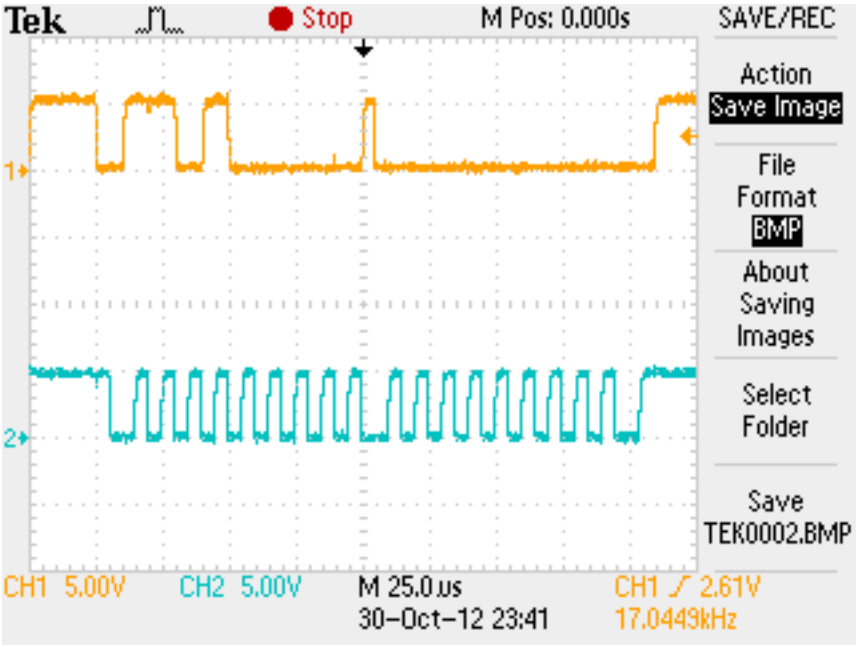


FIG. I.1. : Écriture d'un 0 à l'oscilloscope (original)

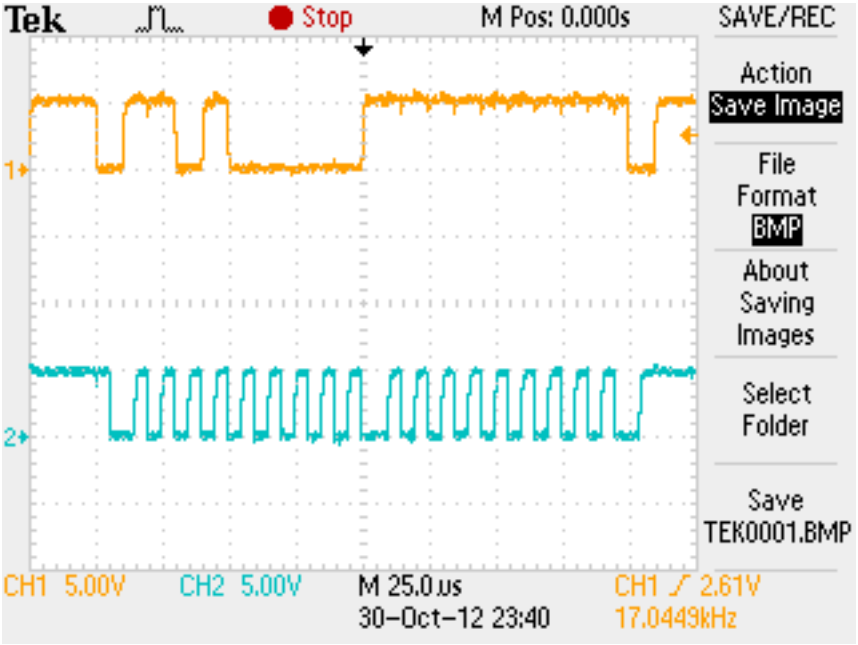


FIG. I.2. : Écriture d'un 255 à l'oscilloscope (original)

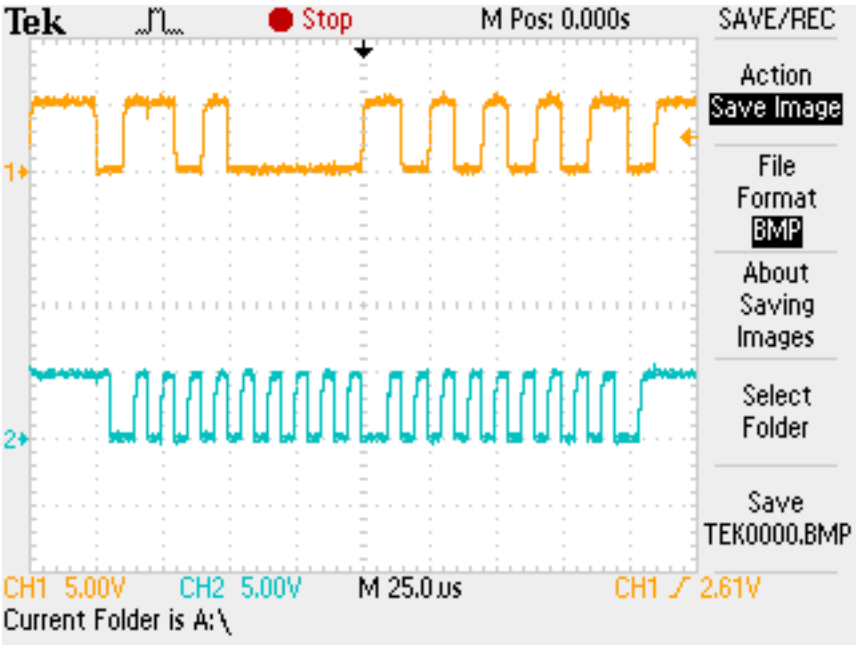


FIG. I.3. : Écriture d'un 170 à l'oscilloscope (original)

J. Graphiques

Voici les graphiques des autres essais réalisés. À cause de leur taille, le premier est à la page suivante.

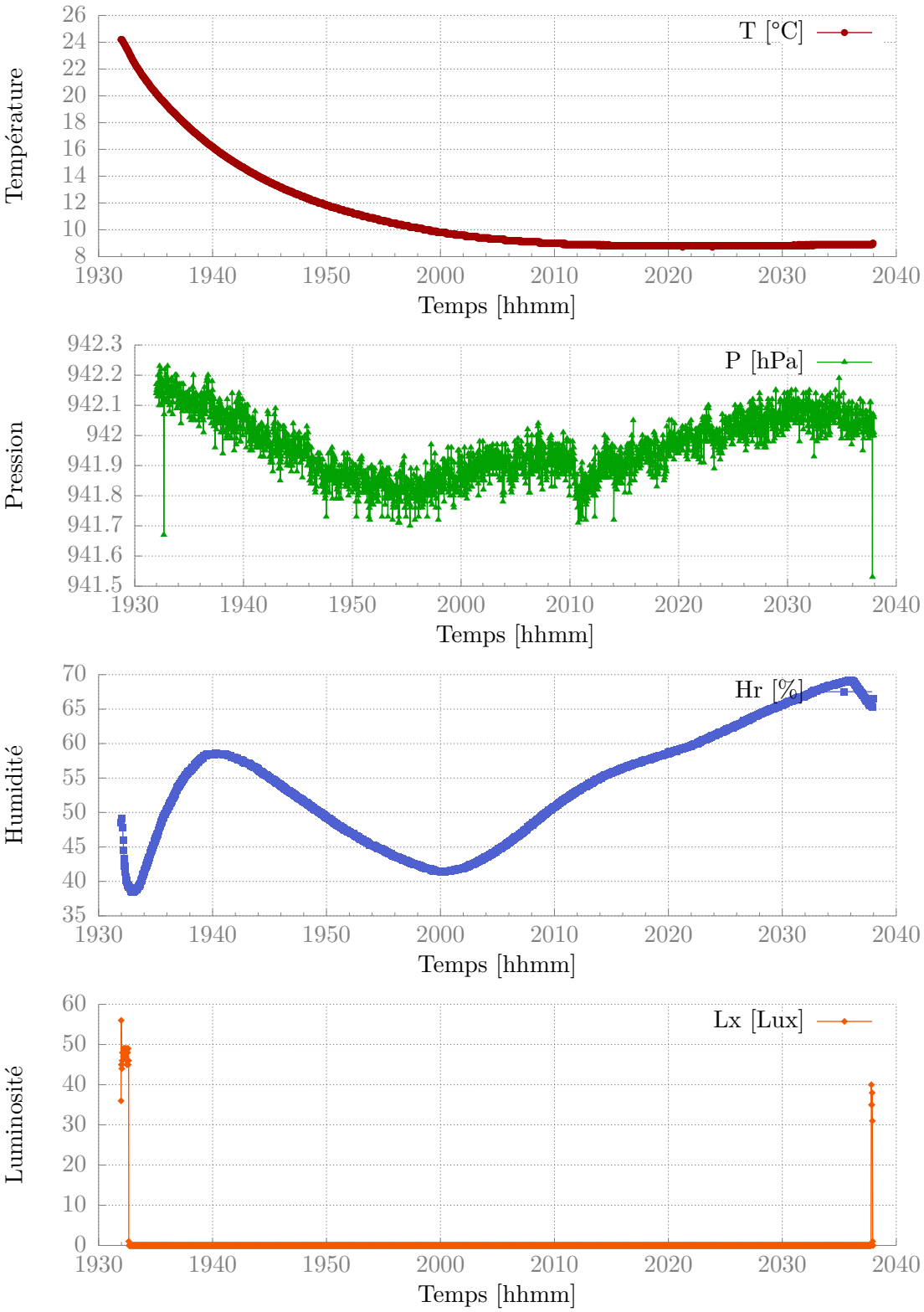


FIG. J.1. : Premier essai dans le frigo

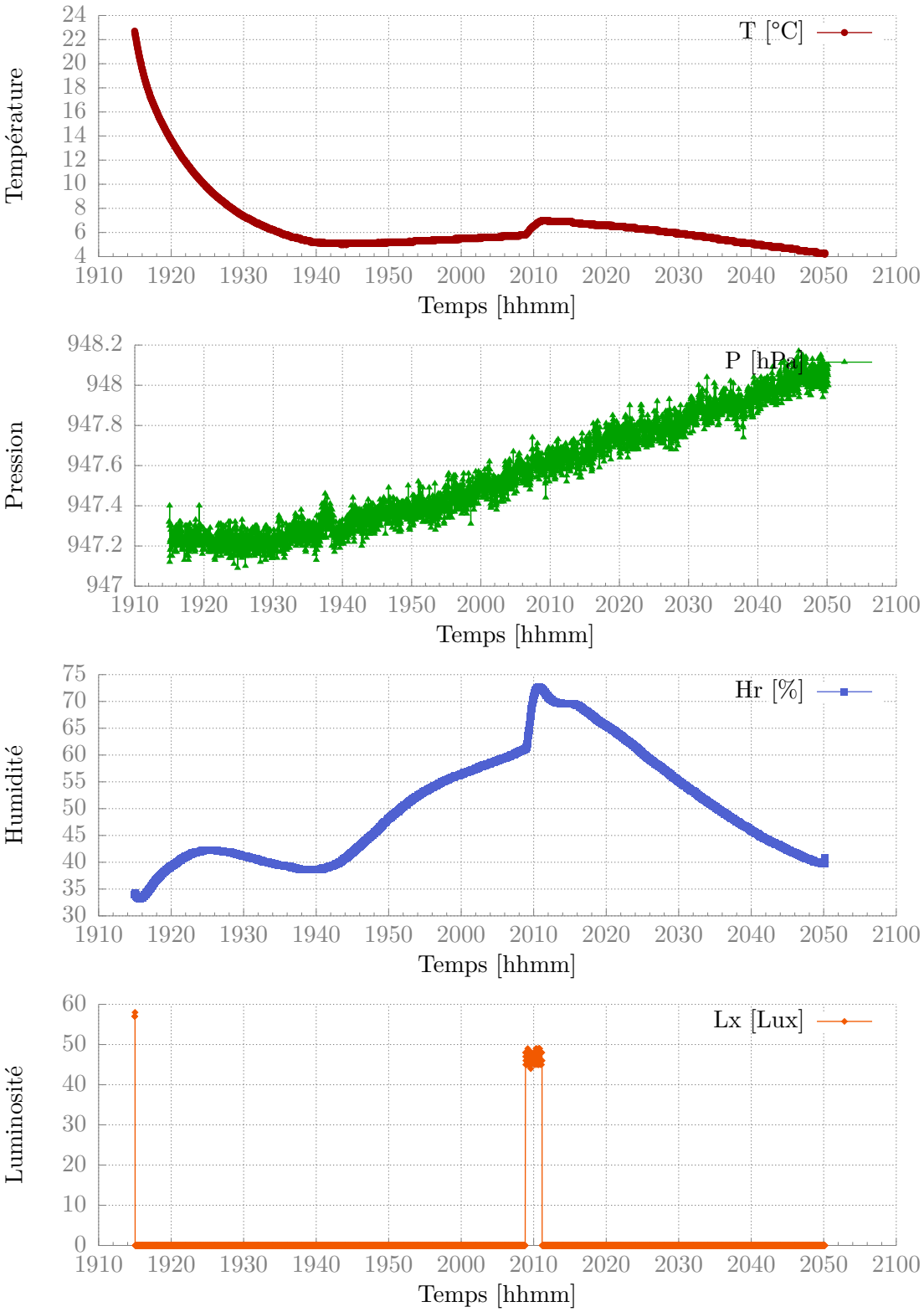


FIG. J.2. : Deuxième essai dans le frigo