GPA788 – Communication I2C entre un coordonnateur et deux nœuds

Philippe Boivin
Étudiant en génie de la production automatisée
École de technologie supérieure
1100 Notre-Dame Ouest,
Montréal, Canada
philippe.boivin.2@ens.etsmtl.ca

Sandrine Bouchard
Étudiante en génie de la production automatisée
École de technologie supérieure
1100 Notre-Dame Ouest,
Montréal, Canada
sandrine.bouchard.1@ens.etsmtl.ca

Alexandre Lins-d'Auteuil
Étudiant en génie de la production automatisée
École de technologie supérieure
1100 Notre-Dame Ouest,
Montréal, Canada
alexandre.lins-dauteuil.1@ens.etsmtl.ca

Benedikt Franz Witteler
Étudiant en génie de la production automatisée
École de technologie supérieure
1100 Notre-Dame Ouest,
Montréal, Canada
benedikt-franz.witteler.1@ens.etsmtl.ca

***Résumé*—Cet article traite de la mise en œuvre d'un coordonnateur pour faire la gestion de deux nœuds sur lesquels sont branchés des capteurs. La fonction du coordonnateur est d'assurer une communication avec chaque nœud et de récolter les données acquises par chaque capteur. Le protocole de communication utilisé, ainsi que le processus de conception des nœuds et du coordonnateur sont expliqués. Finalement, les résultats obtenus sont dévoilés.**

I. Introduction

L'acquisition de diverses données à partir de plusieurs capteurs différents est une pratique bien commune dans l'industrie. Effectivement, il est plutôt rare que l'on analyse un procédé ou un phénomène à l'aide d'une seule variable du procédé. Cela nous mène donc à l'analyse du procédé à partir de plusieurs variables. Ainsi, nous utilisons plusieurs capteurs qui ont chacun pour rôle l'acquisition des données d'une variable du procédé.

Le stockage des données sur un périphérique local est très peu pratique. Le recours aux services infonuagiques devient donc une pratique courante au moyen de l'internet des objets. Avec plusieurs capteurs, il faut cependant établir une connexion entre chacun d'eux et le service infonuagique, rendant ainsi le processus laborieux et très peu flexible. En effet, il faudrait gérer indépendamment chacun des capteurs utilisés pour analyser le procédé ou le phénomène. Le recours à un coordonnateur devient une solution viable pour centraliser la gestion des capteurs et les échanges de données avec le service infonuagique.

L'objectif principal visé consiste donc à mettre en œuvre un coordonnateur qui assurera la gestion des deux nœuds intelligents présentés dans l'article précédent. Ce coordonnateur utilisera le protocole de communication I2C pour établir l'échange des données avec les nœuds, et un point d'accès WIFI pour transmettre les données à l'ordinateur. La communication avec le service infonuagique sera présentée dans un article ultérieur.

II. Bus de communication I2C

Tout d'abord, le protocole de communication utilisé dans ce projet est le protocole I2C (*Inter-integrated Circuit*). Il s'agit d'une communication série synchrone qui requiert seulement deux fils et une masse.

Le premier fil, le SDA (Serial Data Line), sert à transférer les données du coordonnateur aux nœuds et vice-versa. Le second fil, le SCL (Serial Clock Line), permet la synchronisation des envois de données entre les deux composantes.

L'utilisation spécifique à ce projet ne requiert qu'un seul coordonnateur restant actif sur le bus de communication. Celui-ci contrôle le fil SCL et les opérations qui devront être réalisées sur le fil SDA.

Le protocole de transfert de données est représenté par les 2 figures suivantes :

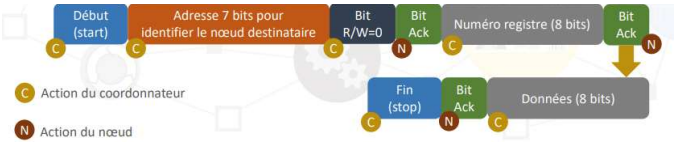

FIGURE 1. : Mode Écriture

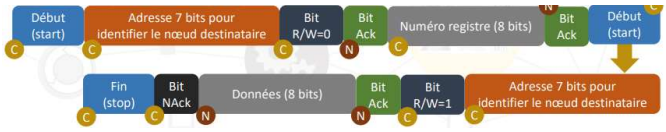

FIGURE 2. : Mode Lecture

Le rôle du coordonnateur dans le projet consiste à permettre le transfert de données acquises par les capteurs branchés sur les deux nœuds vers le coordonnateur. Il est aussi possible pour le coordonnateur d'envoyer différentes commandes aux nœuds sur le bus I2C. Les données sont par la suite transférées du coordonnateur vers l'ordinateur par communication Wi-Fi.

III. CONCEPTION DES NOEUDS ET DU COORDONATEUR

Dans la communication I2C, un nœud est la composante qui exécute et retourne des données selon les commandes du coordonnateur.

Dans ce projet, le coordonnateur et le nœud utilisent l'ordre *Little Endian* pour le stockage en mémoire des données, ce qui détermine l'ordre des octets reçus et envoyés entre l'Arduino et le Pi.

A. Nœud DHT11

Deux fonctionnalités sont implémentées sur l'Arduino du nœud DHT11, la mesure de la température et de l'humidité de l'environnement et la communication I2C avec un coordonnateur. La mesure est faite par l'utilisation de la librairie dhtlib_gpa788 présentée dans un article précédent. La communication est mise en œuvre comme suit.

Puisque la communication I2C manipule des bits, l'organisation I2C du nœud se présente sous la forme d'une carte de registre. Celle-ci permet de traduire des données de plusieurs types en octet afin que le nœud puisse les transférer à l'aide du port I2C vers le coordonnateur. Cette carte de registre est faite par le biais d'une union dans le langage C.

Cette union est composée de 2 parties : une structure de données et un tableau de X nombre d'octets. Son rôle consiste à permettre l'écriture des données dans les variables contenues dans la partie et de les lire à partir du tableau.

Pour le nœud DHT11, voici les variables contenues dans la structure qui seront envoyées vers le coordonnateur sous forme d'un tableau:

- Ts (uint8_t): taux d'échantillonnage (1 octet)
- nb_echantillons (int16_t) : nombre d'échantillons (2 octets)
- temperature (float) : température mesurée par le DHT11 en Celsius (4 octets)
- humidite (float) : humidité mesurée par le DHT11 (4 octets)

La librairie Wire[1] permet de simplifier la gestion des interruptions et de réaliser la communication I2C à l'aide de 3 simples fonctions.

Wire.begin( ) : cette fonction permet d'initialiser la bibliothèque Wire avec l'adresse du nœud afin de faire la communication I2C.

Wire.onReceive( ) : cette fonction permet de gérer une commande ou une donnée venant du coordonnateur.

Wire.onRequest( ) : cette fonction permet de gérer l'envoi des données vers le coordonnateur.

B. Nœud Electret-MAX4466

Sur l'Arduino du nœud Electret, le niveau sonore équivalent L_{eq} est mesuré à l'aide de la librairie Calculateur_Leq présentée dans l'article précédent. Le L_{eq} est ensuite envoyé au coordonnateur par I2C.

Tout comme pour le nœud DHT11, la gestion des données dans le nœud Electret est facilitée en utilisant une union. Celle-ci contient une structure pour accéder aux données depuis le code source C et un tableau pour le transfert des données par le port I2C.

Les données traitées par le nœud Electret sont les suivantes :

- Ts (uint8_t) : taux d'échantillonnage (1 octet)
- nb_echantillons (int16_t) : nombre d'échantillons (2 octets)
- Leq (float) : valeur calculée par l'Arduino à partir des mesures du capteur (4 octets)

Afin de réaliser le nœud Electret, les mêmes fonctions de la librairie Wire sont utilisées que pour l'implémentation du DHT11 :

Wire.begin( )

Wire.onReceive( )

Wire.onRequest( )

Ces fonctions sont utilisées exactement de la même façon. Par contre, elles traitent du niveau sonore équivalent capturé par le nœud au lieu de la température et de l'humidité.

Contrairement à la classe dhtlib_gpa788, le Calculateur_Leq n'effectue pas la gestion du temps pour les mesures du capteur sonore dans la classe elle-même. Afin de mesurer le niveau sonore équivalent, il est nécessaire d'appeler les fonctions Accumulate( ) et Compute( ) dans une boucle infinie. Cette boucle est répétée constamment lorsque le nœud a été activé par la commande « GO ». Après la période d'échantillonnage défini pour ce nœud, L_{eq} est lu et stocké dans une variable avant d'être envoyé au coordonnateur lorsqu'il en fait la demande.

C. Coordonnateur

Dans le coordonnateur, la bibliothèque SMBus est utilisée pour instancier un objet de type SMBus lié au port I2C1. Ce type d'objet permet de définir un temps de transition entre les valeurs 0 et 1.

Dans ce projet, l'objet bus servira à envoyer des requêtes aux nœuds et à lire les valeurs envoyées par les nœuds en utilisant les fonctions suivantes.

Write_byte( ) : cette fonction permet d'envoyer une valeur à l'adresse I2C spécifiée.

Write_i2c_bloc_data( ) : cette fonction permet de transmettre une commande et des données en bloc à l'adresse I2C spécifiée.

Les données doivent obligatoirement envoyées sous forme de liste.

Read_byte( ) : cette fonction permet de lire les valeurs envoyées à l'adresse I2C spécifiée.

La méthode d'adressage *Little Endian* utilisée dans ce projet requiert un ordre spécifique de lecture des valeurs. Les 4 octets nécessaires pour exprimer les mesures enregistrées par les capteurs doivent donc être placés dans un vecteur dans l'ordre inverse de celui envoyé par les nœuds. Ce vecteur de 4 octets est ensuite converti en nombre à virgule flottante (float) grâce à la fonction unpack de la bibliothèque struct.

Le coordonnateur sert essentiellement à envoyer des requêtes aux différents nœuds et lire les données que ces nœuds envoient au moyen des commandes suivantes.

Send_stop( ) : cette fonction envoie la commande d'arrêt (0xA1) sur le bus I2C à l'adresse spécifiée.

Send_go( ) : cette fonction envoie la commande de démarrage (0xA2) sur le bus I2C à l'adresse spécifiée.

Send_Ts( ) : cette fonction envoie la commande de changement de période d'échantillonnage (0xA0), ainsi que la nouvelle valeur choisie pour cette période sur le bus I2C à l'adresse spécifiée. Cette valeur doit être exprimée en seconde.

Send_reset( ) : cette fonction envoie la commande de redémarrage (0xA3) sur le bus I2C à l'adresse spécifiée.

Send_pause( ) : cette fonction envoie la commande de pause (0xA4) ainsi que la durée choisie pour la pause sur le bus I2C à l'adresse spécifiée. La durée doit être exprimée en seconde.

Read_Temp( ) : cette fonction lit la température du nœud à l'adresse spécifiée.

Read_Hum( ) : cette fonction lit l'humidité du nœud à l'adresse spécifiée.

Read_Leq( ) : cette fonction lit le niveau sonore équivalent du nœud à l'adresse spécifiée.

Read_SNumber( ) : cette fonction lit le numéro de l'échantillon de la mesure du nœud à l'adresse spécifiée.

Les exceptions possibles lors de l'utilisation de l'ensemble de ces fonctions sont CoordException et IOError. Les fonctions de lecture peuvent aussi admettre l'exception error de la bibliothèque struct. Toutes ces erreurs sont capturées et traitées de la façon typique de Python.

Pour transmettre les commandes, il faut tout d'abord instancier un objet de type SMBus. Pour faire bonne mesure, il est préférable d'arrêter les nœuds à l'aide de la commande « STOP » avant de commencer à envoyer d'autres demandes. Le temps d'échantillonnage est ensuite réglé à 6 secondes et la commande « GO » est envoyée. Chacune de ces commandes est accompagnée de l'affichage d'une phrase indiquant à l'utilisateur où en est le programme.

Finalement, le coordonnateur entre dans une boucle infinie qui lit le temps, l'humidité, le niveau sonore équivalent et le nombre d'échantillon envoyés par les nœuds à toutes les 15 secondes. Ces données, accompagnées de l'horodatage, sont ensuite affichées sur le terminal du Pi. Afin de tester les commandes « PAUSE » et « RESET », celles-ci sont envoyées aux nœuds après 10 à 12 échantillons et 15 échantillons respectivement.

IV. Résultats

Afin de valider le bon fonctionnement de nos capteurs et du bus de communication I2C, nous avons effectué des essais d'échantillonnage et tenté de confirmer la validité et la concordance des données obtenues.

Pour faire le diagnostic de notre système, nous avons utilisé un ordinateur branché aux 2 nœuds, puis un autre ordinateur en communication Wi-Fi avec le coordonnateur. De ce fait, nous étions en mesure d'afficher de l'information sur le terminal série des nœuds respectif ainsi que la console Python du coordonnateur. Afin de valider l'exactitude de la communication I2C entre les deux, l'observation des terminaux des nœuds et du coordonnateurs est essentielle.

A. Nœud DHT11

Lors de l'envoi d'une commande « GO », « STOP », « PAUSE » ou « RESET, le nœud DHT11 répond avec succès. En effet, lorsque la réception de la commande « GO » s'effectue avec succès, un message s'affiche sur le terminal de l'ordinateur branché sur le nœud. Les données de chaque échantillon sont alors imprimées sur le terminal série à la fin de chaque période d'échantillonnage.

Lors de la réception de la commande « PAUSE » et de la valeur du temps de pause, un message s'affiche sur le terminal de l'ordinateur branché sur le nœud indiquant dans combien de temps le nœud redémarrera de façon automatique. Lors de la réception de la commande « STOP », un message s'affiche sur le terminal de l'ordinateur branché sur le nœud, l'échantillonnage est arrêté et les valeurs ne sont plus affichées.

Lors de la réception de la commande « RESET », un message s'affiche sur le terminal de l'ordinateur branché sur le nœud. Cette commande ne change pas la commande précédente envoyée par le coordonnateur, c'est-à-dire que si la commande précédente envoyée était « GO », le nœud continue d'afficher les valeurs de façon périodique, mais le nombre d'échantillons est remis à zéro et les dernières valeurs lues par le DHT11 sont effacées. Si la commande précédente envoyée était « STOP », le nœud reste à l'arrêt à la suite de la commande « RESET ».

Lorsque le coordonnateur demande la valeur de la température ou de l'humidité au nœud, la valeur est bel et bien retournée au coordonnateur, et la valeur retournée est affichée sur le terminal de l'ordinateur en communication Wi-Fi avec le coordonnateur. Nous avons vérifié la concordance entre les valeurs lues sur le terminal du nœud et sur le terminal du coordonnateur.

Nous avons aussi testé la commande de changement de la période d'échantillonnage du nœud. Lors de l'envoi de la commande, les données affichées sur le terminal de l'ordinateur reflétaient bel et bien le changement de la période.

Finalement, lorsque qu'une commande erronée est envoyée au nœud, ce dernier ne répond simplement pas et continue l'action en cours. Si le coordonnateur envoie 3 octets ou plus au nœud, un message d'erreur est affiché sur le terminal de l'ordinateur connecté au nœud indiquant que cette fonction n'est pas prise en charge.

B. Nœud Electret-MAX4466

Lors de l'envoi d'une commande « GO », « STOP », « PAUSE » ou « RESET », le nœud Electret répond avec succès. En effet, lorsque la réception de la commande « GO » s'effectue avec succès, un message s'affiche sur le terminal de l'ordinateur branché sur le nœud. Les données de chaque échantillon sont alors imprimées sur le terminal série à la fin de chaque période d'échantillonnage.

Tout comme le nœud DHT11, lors de la réception de la commande « PAUSE », « STOP » ou « RESET », le comportement du nœud correspond exactement aux attentes. Effectivement, ce nœud montre les mêmes réponses aux commandes que le nœud DHT11.

Lorsque le coordonnateur demande la valeur du niveau de bruit équivalent au nœud, la valeur est bel et bien retournée au coordonnateur et la valeur retournée est affichée sur le terminal de l'ordinateur en communication Wi-Fi avec le coordonnateur. Nous avons vérifié la concordance entre les valeurs lues sur le terminal du nœud et sur le terminal du coordonnateur.

Tout comme avec le nœud DHT11, nous avons testé la commande de changement de la période d'échantillonnage du nœud avec succès. Finalement, les commandes erronées ne sont pas prises en compte et le nœud indique si le nombre d'octets reçu dépasse la quantité prévue.

C. Coordonnateur

Lors du démarrage du coordonnateur, un message est affiché sur l'ordinateur en connexion sans fil avec ce dernier, montrant que l'opération s'est effectuée avec succès. De plus, la période d'acquisition des données définie dans le fichier de constantes du coordonnateur est réglée correctement, puis l'acquisition des données débute.

Par la suite, les données suivantes sont affichées à intervalles réguliers et pour chaque nœud : l'adresse du nœud interrogé, le nombre d'échantillons depuis le début de l'acquisition ou de la dernière commande « RESET » envoyée, puis la ou les valeurs lues par les capteurs (température et humidité, ou niveau de bruit équivalent).

Lorsque des erreurs surviennent, des messages sont affichés sur le terminal série de l'ordinateur. Par exemple, lorsqu'une erreur se produit sur le bus de communication I2C, un tel message est affiché avec le code d'erreur correspondant. De plus, si une erreur de conversion des données se produit, un message similaire apparait. Finalement, si une erreur se produit au niveau des fonctions de communication avec les nœuds, le message d'erreur correspondant est affiché sur le terminal de l'ordinateur.

Nous avons manuellement créé des conditions d'erreur dans le code pour vérifier le bon fonctionnement du gestionnaire d'erreur.

Lors de l'arrêt du programme du coordonnateur par l'envoi de la commande CTRL+C sur le terminal série de l'ordinateur connecté au coordonnateur, l'échantillonnage des nœuds est bel et bien arrêté. La réception de la commande est effectivement affichée sur les terminaux des deux nœuds.

V. CONCLUSION

La réalisation et l'intégration du coordonnateur pour la gestion des deux nœuds conçus précédemment s'est révélée être un succès.

Effectivement, les commandes envoyées par le coordonnateur sont correctement reçues par chacun des nœuds qui interprètent les commandes adéquatement. De plus, la gestion des erreurs de communication demeure bien effectuée par chacun des éléments du système.

En ce qui a trait aux données reçues par le coordonnateur, elles concordent avec les données affichées sur le terminal série de l'ordinateur branché sur chaque nœud, ce qui prouve que le transfert des données s'effectue avec succès.

La prochaine étape du développement de notre système d'acquisition de données à partir de nos capteurs intelligent est le transfert de ces données vers un service de stockage infonuagique qui permettra de construire une base de données ainsi que faire l'apprentissage automatique sur ces dernières.

Sources

[1] Référence Arduino français, Librairie Wire, repéré à http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieWire