

Rapport de projet piste

Approche multi-capteurs pour la gestion
environnementale d'un bassin versant

Réalisé par :

Belhadj Mohamed Nahed

Hedhly Wafa

Encadré par :

Cherif Soufiane

Choubani Fethi

Année universitaire : 2016/2017

Avant-Propos

Ce projet est une partie intégrante du projet PISTE dispensé aux étudiants INDP2 de la dominante SYSTEL. L'objectif principal est de développer l'initiative de l'étudiant et améliorer ses compétences pour le travail en équipe.

Durant ce projet, on a développé notre compétence, premièrement, dans le domaine internet of things plus précisément dans le domaine de programmation des capteurs communiquant entre eux et, deuxièmement on se familiariser à l'environnement de l'entreprise avec ses différentes divisions.

Ce rapport porte sur la présentation du projet global, principalement notre sous-projet, ainsi que la description des différentes tâches effectuées.

Remerciement

C'est avec un réel plaisir que nous résumons ces quelques lignes en signe de gratitude et de profonde reconnaissance à tous ceux qui de près ou de loin ont participé à la réalisation et l'accomplissement de ce travail et en nous apportant le soutien moral, intellectuel et technique dont nous avons besoin.

Nous tenons, en premier lieu, de remercier nos encadrants, M. Soufiane Cherif et M. Fethi Chobani pour ses conseils constructifs et judicieux. Nous les remercions également de leurs aides, leur patience pour nous mener à bien rédiger ce rapport.

Il nous est également fort gré de remercier les doctorantes, Mlle Maymouna, Mlle Asma, Mlle Zakiya, Mlle Leila pour leurs bien nombreuses contributions à l'accomplissement de ce travail. Nos remerciements sont adressés aussi à tous ceux qui nous ont accompagnées tout au long de ce projet avec beaucoup de patience et de pédagogie, et qui ont eu la gentillesse de faire de cette expérience professionnelle un moment très profitable.

Table des matières

Table des figures	IV
Introduction générale	1
Chapitre 1 : Réseau de capteurs sans fil	3
Introduction.....	4
I. Réseau de capteurs sans fil	4
II. Topologie des Réseaux de capteurs	4
III. Architecture d'un micro-capteur.....	5
IV. Les protocoles de communication pour les réseaux de capteurs.....	6
V. Zolertia	7
Conclusion	7
Chapitre 2: Tâches effectuées	8
Introduction.....	9
Tâche 1 : Spécification des capteurs.....	9
Tâche 2 : Installations	10
Tâche 3 : Simulations sur Cooja	10
Tâche 4 : Apprendre la Structure d'un programme en Contiki	11
Tâche 5 : lecture de la température et de l'accélération depuis un capteur Zolertia...	12
Tâche 6 : Lecture des données depuis les capteurs Phidgets	14
Tâche 7 : Communication entre deux nœuds.....	14
Tâche 8 : Communication entre 3 zolertia.....	22
Conclusion	24
Conclusion générale.....	25
Webographie.....	26
Annexes	27

Table des figures

Figure 1 : Décomposition du projet global	2
Figure 2 : Vue globale du projet	2
Figure 4 : Architecture simplifiée d'un micro-capteur.....	6
Figure 5 : Comparaison des protocoles sans fil	7
Figure 6 : Zolertia	7
Figure 7 : Caractéristiques de mote Z1	7
Figure 9 : Exemple de simulation sur LabVIEW	10
Figure 10 : Simulation du programme broadcast-exemple.c.....	10
Figure 11 : Instruction de lecture de la température depuis z1	13
Figure 12 : Résultat de l'implémentation du programme test-TMP102.c.....	13
Figure 13 : Instructions de lecture de l'accélération linéique depuis z1	13
Figure 14 : Résultat de l'implémentation du programme test-adxl345.c	13
Figure 15 : Programme Phidget-temp.c	14
Figure 16 : Fonction receiver.....	16
Figure 17 : Fonction set-global-address.....	16
Figure 18 : Process Thread	17
Figure 19 : Fonction receiver.....	18
Figure 20 : Fonction create-rpl-dag	19
Figure 21 : Process Thread	19
Figure 22 : Résultat de l'implémentation des codes Sender et Receiver sur deux noeuds z1.....	20
Figure 23 : Partie du code différente pour l'accélération au niveau du Sender	21
Figure 24 : Fonction receiver.....	21
Figure 25 : Résultat de l'implémentation	22
Figure 26 : Ports de communication	23
Figure 27 : Fonction receiver pour le Sender1	23
Figure 28 : Fonction receiver pour le Sender2	23
Figure 29 : Appel de la fonction simple-udp-register.....	24

Introduction générale

Le suivi multi-temporel de la variation des paramètres environnementaux (topographie, nature de sol, lac d'eau, température, pression, humidité, ...) est fondamental pour comprendre et surtout gérer l'impact des changements climatiques que subit notre planète. En particulier la ressource en eau qui devient de plus en plus rare nécessite un suivi particulier et l'adoption de nouvelles technologies et stratégies pour la préserver. Parmi les différents moyens d'accès à la ressource en eau on trouve les réservoirs à surface libre (lacs naturel ou de retenue, bassin versant). La télédétection, entre autre, les réseaux de capteurs sans fil, paraît le potentiel le plus évident pour les gérer.

En effet, les réseaux de capteurs, sont de plus en plus utilisés dans la gestion de l'environnement grâce notamment aux derniers développements réalisés dans le domaine des technologies sans-fils. Depuis quelques années, le besoin d'observer, d'analyser et de contrôler des phénomènes physiques sur des zones étendues est essentiel pour de nombreuses applications environnementales et scientifiques. Cette nouvelle manière d'envisager la métrologie, en détectant un phénomène à différents points disséminés sur un système ou un site, fait émerger de nouvelles problématiques technologiques, par exemple sur l'autonomie énergétique des capteurs, et de nouveaux types d'applications nous permettant de mieux connaître notre environnement et d'anticiper les problèmes de sécurité, de pollution, de risques naturels, de défaillances, de maintenances, ou plus généralement de tout phénomène non désiré qui pourrait être anticipé.

Dans le cadre du projet PISTE, on a choisi de suivre les paramètres environnementaux au niveau du bassin versant de Kamech (263 ha) situé sur le Cap Bon en Tunisie via un réseau de capteurs sans fil et un drone. Le but global de ce projet est de réaliser un Dashboard permettant de fournir une carte du site d'étude sur laquelle sont géo-localisés les différents capteurs utilisés et la position du drone.

Ainsi, le projet PISTE est un modèle de projets reposant sur la télédétection comme fonctionnalité de base pour la surveillance à distance d'une source d'eau dans une région qui connaît actuellement une forte intensification des activités agricoles.

Le projet est divisé en des sous-tâches indépendantes entre les 6 binômes.

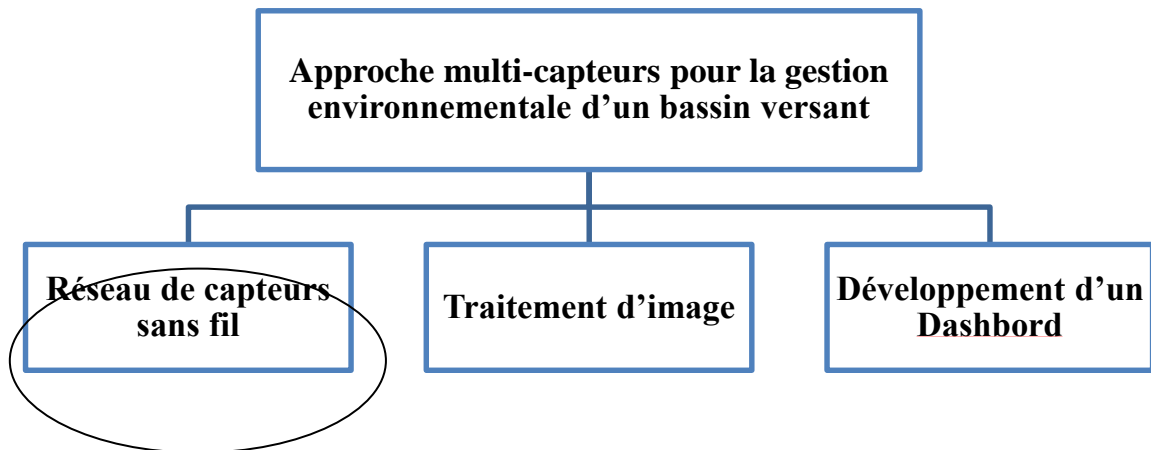


Figure 1 : Décomposition du projet global

Notre sous-projet consiste à récupérer les mesures (température, pression,...) effectuées par les capteurs placés convenablement sur le site et de les envoyer au capteur central. Et pour y arriver, on a eu besoin d'un environnement de travail (matériel et logiciel) spécifique et qui va-t-être détaillé ultérieurement.

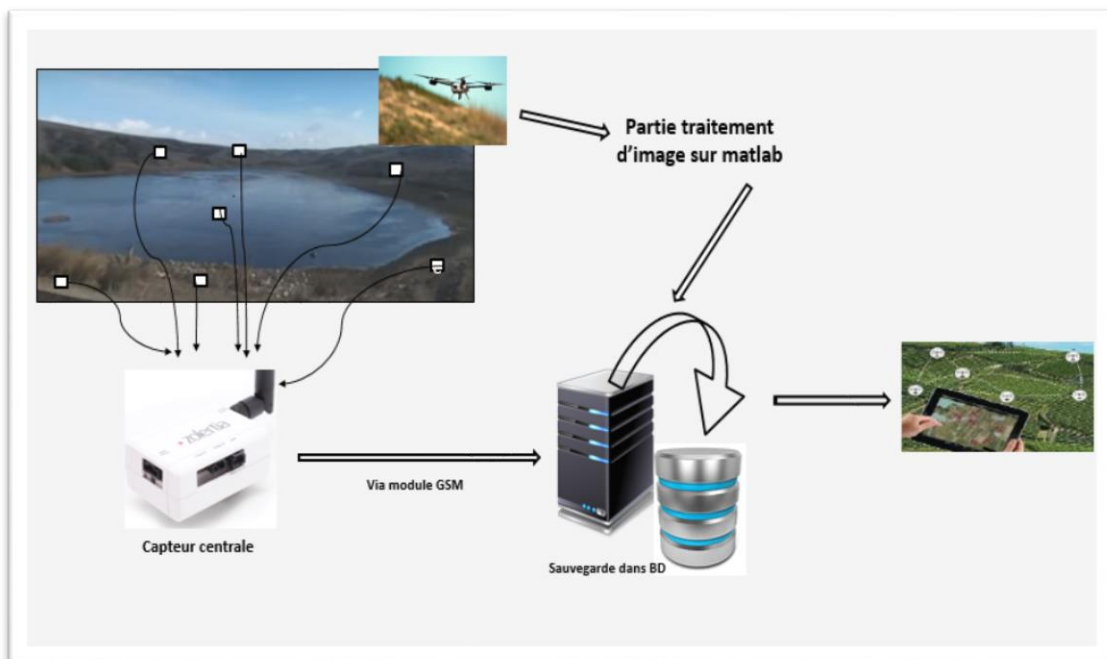


Figure 2 : Vue globale du projet

Chapitre 1 :

Réseau de capteurs sans fil

Introduction

Dans ce chapitre on va expliquer le concept de réseau de capteurs sans fils, ses principes, ses topologies, ses protocoles,...ainsi que la description de la solution technique qu'on a jugé adéquate pour l'accomplissement de ce projet.

I. Réseau de capteurs sans fil

Définition

Les réseaux de capteurs sont des systèmes qui regroupent plusieurs capteurs autonomes, interconnectés par des liaisons sans fils afin de couvrir une zone cible. Ces capteurs peuvent interagir entre eux et avec un système externe (par exemple Internet) par le biais de communication sans fil ou filaire. Les réseaux de capteurs sans-fils concentrent les dernières avancées technologiques et représentent l'opportunité de nouvelles applications. En Anglais, on parle de « WSN » pour « Wireless Sensor Networks ».[1]

Leur but principal est de surveiller des endroits peu accessibles à moindre coût. Ils sont utilisés dans plusieurs domaines (environnemental, commercial, militaire,...).

Principe

Les capteurs sans fil communiquent par le biais des ondes radioélectriques. Chaque capteur peut transmettre une information indépendamment ou avec l'aide des autres capteurs vers une « station de base » capable de transmettre l'information à l'utilisateur final, par le biais d'Internet ou d'un réseau télécom GSM dans la majorité des cas. Les capteurs sont capables de mesurer des grandeurs physiques, chimiques ou biologiques, de traiter ces informations et de les stocker. Ils sont alimentés électriquement via une batterie individuelle optimisée pour des tâches comme le traitement de l'information et la communication.

II. Topologie des Réseaux de capteurs

Il existe plusieurs topologies pour les réseaux à communication radio. Nous discutons ci-dessous des topologies applicables aux réseaux de capteurs [2] :

La Topologie en étoile

Dans cette topologie un nœud central envoie ou reçoit un message via un certain nombre de nœuds. Ces nœuds peuvent seulement envoyer ou recevoir un message de l'unique nœud central, il ne leur est pas permis de s'échanger des messages.

La topologie maillée (Mesh Network)

Dans ce cas (dit « *communication multi-sauts* »), tout nœud peut échanger avec n'importe quel autre nœud du réseau. Un nœud voulant transmettre un message à un autre nœud hors de sa portée de transmission, peut utiliser un nœud intermédiaire pour envoyer son message au nœud destinataire.

La topologie en arbre (Cluster Network)

Un réseau arbre est composé d'une « route principale » par laquelle transitent toutes les informations échangées entre les capteurs sans fil.

Ce type de réseau est constitué de la façon suivante :

Des capteurs sans fil qui enregistrent et transmettent les informations des capteurs.

Des routeurs sans fil qui constituent la route principale d'échange d'informations.

Un coordinateur du réseau de capteur sans fil qui organise et transmet les informations en provenance du réseau sans fil vers l'application de supervision du réseau sans fil.

III. Architecture d'un micro-capteur

Un capteur est composé de 3 unités :

- **Unité d'acquisition** : elle est composée d'un capteur qui va obtenir des mesures numériques sur les paramètres environnementaux et d'un convertisseur Analogique/Numérique qui va convertir l'information relevée et la transmettre à l'unité de traitement.
- **Unité de traitement** : elle est composée de deux interfaces, une interface pour l'unité d'acquisition et une interface pour l'unité de transmission. Cette unité est également composée d'un processeur et d'un système d'exploitation spécifique. Elle acquiert les informations en provenance de l'unité d'acquisition et les envoie à l'unité de transmission.
- **Unité de transmission** : elle est responsable de toutes les émissions et réceptions de données via un support de communication radio.

Ces trois unités sont alimentées par une batterie comme la montre la figure ci-dessous :

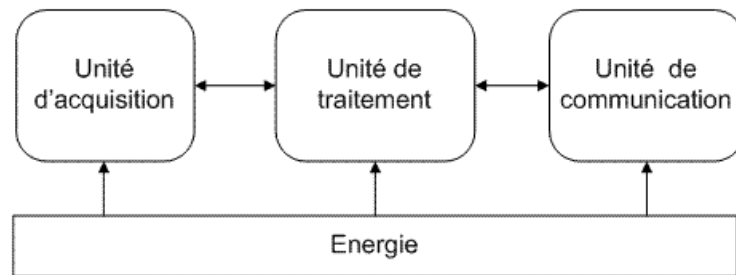


Figure 3 : Architecture simplifiée d'un micro-capteur

IV. Les protocoles de communication pour les réseaux de capteurs

Wifi

C'est une technologie de réseau informatique sans fil mise en place pour fonctionner en réseau internet et, depuis, devenue un moyen d'accès à haut débit à Internet. Il est basé sur la norme IEEE 802.11 (ISO/CEI 8802-11).

ZigBee

C'est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN). Zigbee, est une norme de transmission des données sans fil permettant la communication de machine à machine. Toutes les caractéristiques de ce protocole sont bien adaptées aux systèmes embarqués :

- Faibles besoins en mémoires.
- Une durée de vie très importante de l'ordre de plusieurs années.
- Un très large nombre de nœuds à supporter dans son réseau.
- Ce protocole convient parfaitement aux applications nécessitant une faible vitesse de transfert.

Bluetooth

C'est un standard de communication permettant l'échange bidirectionnel de données, il utilise une technologie radio courte distance destinée à simplifier les connexions entre les appareils électroniques.

Protocole	Zigbee	Bluetooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g
Besoins mémoire	4-32 Kb	250 Kb +	1 Mb +
Durée de vie	Années	Jours	Heures
Nombre de nœuds	65 000+	7	32
Vitesse de transfert	250 Kb/s	1 Mb/s	11-54 Mb/s
Portée	100 m	10 m	100 m

Figure 4 : Comparaison des protocoles sans fil

V. Zolertia

Le module Z1 est une plate-forme générale de développement pour les réseaux de capteurs sans fil (WSN) formé par :

- D'une unité de mesure
- D'un micro-processeur
- D'un module de transmission sans fil
- D'une batterie à capacité limitée et partagée



Figure 5 : Zolertia

Le mote Z1 possède deux capteurs intégrés: Le ADXL345 (accéléromètre 3 axes à haute résolution (13 bits), et de faible puissance), Le TMP102 (capteur de température), de plus, deux ports pour les capteurs externes de types Phidgets, un port pour les capteurs externes de type Ziglet, un Micro-USB et une antenne pour la transmission

Plate-forme	ROM	RAM	La fréquence	La plus basse consommation de courant	# radios	Portée maximale Radio
Z1	92KB	8KB	16MHz	3.5-18uA	1	100 mts (0dBi antenne, LOS)

Figure 6 : Caractéristiques de mote Z1

Conclusion

Ce chapitre représente une passerelle pour entamer l'étape de programmation des capteurs sans fils et permet d'éclaircir et d'organiser les étapes à traiter ultérieurement.

Chapitre 2:

Tâches effectuées

Introduction

Dans ce chapitre on va détailler les tâches qu'on a effectuées dans l'ordre, en précisant pour chaque tâche, sa description technique, son implémentation et ses résultats.

Ordonnancement des tâches

1. Spécification des capteurs
2. Installations
3. Simulations avec Cooja
4. Apprendre la structure d'un programme en contiki
5. Lecture des données (température+accélération) depuis un capteur zolertia
6. Lecture des données depuis les capteurs Phidgets
7. Communication entre deux zolertia
8. Communication entre 3 zolertia

Tâche 1 : Spécification des capteurs

Comme première étape on a commencé par extraire les données des capteurs de type Phidgets liés à une carte Arduino en utilisant LabVIEW comme plate-forme de simulation qui est installé sur Windows. LabVIEW est une plate-forme de conception de systèmes de mesure et de contrôle, basée sur un environnement de développement graphique de National Instruments.

Le diagramme de LabVIEW est lié à une interface utilisateur graphique nommée face-avant. Les programmes et sous-programmes sous forme d'icônes sont appelés des instruments virtuels (VI). Chaque VI possède trois composants : un diagramme qui intègre le code graphique, une face-avant personnalisable par l'utilisateur et un panneau de connexions pour les icônes qui sert d'entrées/sorties pour les variables sous forme de fils. Une fois un VI écrit, il devient une icône pour un programme de plus haut niveau et intégré dans le nouveau diagramme.

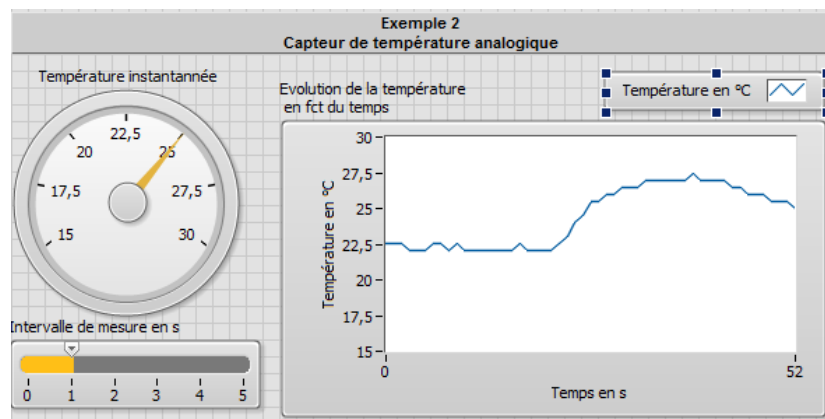


Figure 7 : Exemple de simulation sur LabVIEW

Tâche 2 : Installations

On a commencé tout d'abord par l'installation du système d'exploitation LINUX sur les machines puis on a installé l'environnement de Contiki et le simulateur COOJA. COOJA permet de simuler les connexions réseaux et d'interagir avec les capteurs.

Tâche 3 : Simulations sur Cooja

Le système d'exploitation Contiki englobe plusieurs programmes qui sont définis sous /contiki/examples.

Dans cette étape on a testé sur le simulateur Cooja quelques programmes (unicast-sender, broadcast-sender) liés à la communication entre nœuds z1 qui sont placés sous /contiki/examples/ipv6/simple-udp

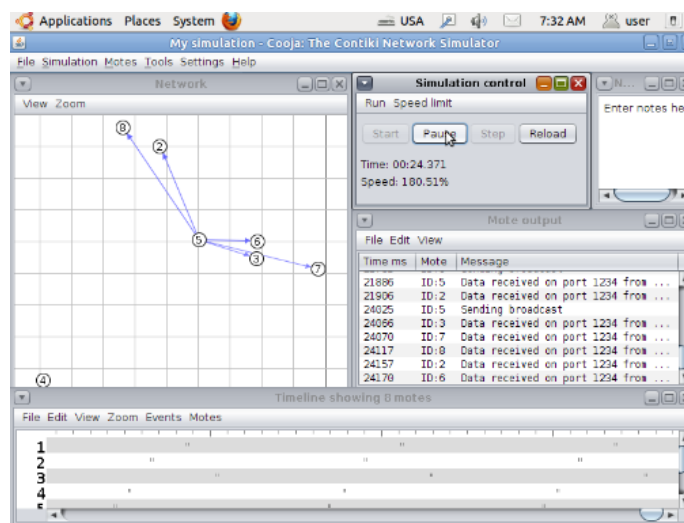


Figure 8 : Simulation du programme broadcast-exemple.c

Tâche 4 : Apprendre la Structure d'un programme en Contiki

Programmer et concevoir des réseaux de capteurs sans fil sur Contiki repose sur la maîtrise des bases de la programmation de ce système d'exploitation, qui propose une bibliothèque spécifique de fonctions et un environnement de développement complet et fonctionnel.

Ainsi, apprendre les bases et les principes de la programmation sur Contiki est indispensable pour arriver par la suite, à rédiger des programmes plus compliqués et sophistiqués.

Voici la structure d'un programme C basique en Contiki qui permet d'afficher une chaîne de caractères sur le terminal [3]:

- 1: **#include "contiki.h"** : contient toutes les déclarations nécessaires aux abstractions de Contiki.
- 2: **#include <stdio.h>** : est déclarée pour pouvoir appeler la fonction *printf()*.
- 3: **PROCESS(my_first_process, "Hello World Process");** : permet de déclarer un nouveau processus Contiki nommé « *Hello World Process* » et ayant comme variable de process *my_first_process*.
- 4: **AUTOSTART_PROCESSES(&my_first_process);** : permet de lancer automatiquement le process *my_first_process* au démarrage du système d'exploitation. Cette méthode peut avoir comme paramètres une liste de process qui vont être lancés automatiquement dans l'ordre.
- 5: **PROCESS_THREAD(my_first_process, ev, data);** c'est la définition de notre process. Le premier paramètre est la variable *my_first_process*, le deuxième *ev* est un évènement, qu'on peut utiliser pour permettre au programme de répondre à des évènements relatifs au système. Le troisième paramètre *data* est une variable de donnée qu'on peut recevoir avec l'évènement.
- 7: **PROCESS_BEGIN();** : déclare le commencement du process'thread.
- 8: **printf("Hello WSN World!\n");** : affiche la chaîne de caractères *Hello WSN World !*

9: PROCESS_END(); : déclare le commencement du process'thread.

Pour compiler le programme, on a besoin d'un fichier *Makefile*, qui est un fichier sans extension, contenant les instructions nécessaires à la compilation. Dans ce fichier, nous devons spécifier la version de Contiki utilisée, les applications (programmes C) prises en charge par ce fichier et inclure le *Makefile* qui existe dans le répertoire du root.

1 : CONTIKI=../.. : spécifie l'emplacement le root (l'administrateur) de Contiki

2 : all : app-name : spécifie les applications (programmes C) à compiler

3 : include \$(CONTIKI)/Makefile.include : permet d'inclure le *Makefile* qui existe dans le répertoire du root

Compilation du programme

Sur le terminal, la compilation et l'implémentation du programme sur un module Zolertia z1 consiste en 3 commandes :

make Target=z1 application.c : spécifier z1 comme plateforme d'exécution du programme.

make Target=z1 savetarget : enregistrer la plateforme d'exécution pour les prochaines applications prises en charge par le même fichier *Makefile*.

make application.upload : compiler et implémenter le programme sur z1 pour l'exécuter.

Tâche 5 : lecture de la température et de l'accélération depuis un capteur Zolertia

La tâche suivante était la lecture de la température captée par le capteur interne de Zolertia, Tmp102. Cette tâche a nécessité un code simple et claire permettant de [5]:

- Initialiser le capteur Tmp102
- Fixer le Timer à un intervalle **TMP102_READ_INTERVAL=CLOCK_SECOND*10**

- Récupérer la valeur de la température mesurée
- Afficher la température

```
PRINTFDEBUG("Reading Temp...\n");
raw = tmp102_read_temp_raw();
```

Figure 9 : Instruction de lecture de la température depuis z1

```
Temp = 25.8750
Temp = 25.9375
Temp = 25.9375
Temp = 26.0000
Temp = 26.0000
Temp = 26.0625
Temp = 26.0625
Temp = 26.1250
Temp = 26.1250
Temp = 26.1250
Temp = 26.1875
Temp = 26.1875
```

Figure 10 : Résultat de l'implémentation du programme test-TMP102.c

Concernant la mesure de l'accélération linéique par le capteur interne ADXL345, le code est plus compliqué et long puisqu'il doit prendre en considération tous les effets exceptionnels pouvant survenir comme la chute libre(Free Fall Detection), et les coups (Tap and Double Tap Detection)...

Dans le Process Thread, le principe reste le même :

- Initialiser le capteur ADXL345
- Gérer les évènements exceptionnels
- Fixer le Timer à un intervalle
ACCM_READ_INTERVAL=CLOCK_SECOND
- Récupérer la valeur de l'accélération linéique sur chaque axe(x, y et z)
- Afficher les mesures

```
x = accm_read_axis(X_AXIS);
y = accm_read_axis(Y_AXIS);
z = accm_read_axis(Z_AXIS);
printf("x: %d y: %d z: %d\n", x, y, z);
```

Figure 11 : Instructions de lecture de l'accélération linéique depuis z1

```
x: 8 y: 2 z: 237
x: 7 y: 2 z: 237
x: 6 y: 1 z: 236
x: 5 y: 3 z: 238
x: 6 y: 2 z: 236
x: 7 y: 2 z: 237
x: 6 y: 3 z: 237
x: -12 y: 13 z: 237
x: 7 y: 1 z: 239
x: 7 y: 2 z: 236
x: -15 y: -16 z: 239
x: 7 y: 2 z: 234
x: 7 y: 1 z: 237
x: 8 y: 2 z: 235
x: 7 y: 3 z: 236
```

Figure 12 : Résultat de l'implémentation du programme test-adxl345.c

Tâche 6 : Lecture des données depuis les capteurs Phidgets

Contiki a fourni le programme C et toutes les bibliothèques nécessaires à la récupération des données depuis les différents capteurs Phidgets (de même pour les capteurs Ziglet). Ainsi, cette tâche est facile à effectuer.



```
*phidget-temp.c x

#include <stdio.h>
#include "contiki.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include "dev/z1-phidgets.h"

/*-----*/
PROCESS(test_button_process, "Test Button & Phidgets");
AUTOSTART_PROCESSES(&test_button_process);
/*-----*/
PROCESS_THREAD(test_button_process, ev, data)
{
    PROCESS_BEGIN();
    SENSORS_ACTIVATE(phidgets);
    SENSORS_ACTIVATE(button_sensor);

    while(1) {
        printf("Please press the User Button\n");
        PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event &&
                                data == &button_sensor);

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        leds_toggle(LED_GREEN);
        printf("Button clicked\n");

        printf("la temperature 5V 2:%d\n", phidgets.value(PHIDGETSV_2));
    }
    PROCESS_END();
}
```

Figure 13 : Programme Phidget-temp.c

Reste à distinguer entre les différents types de mesures (température, humidité, lumière,...), puisque le programme est le même à implémenter. Ceci se fait facilement par de simples formules de conversion de données spécifique à chaque type de grandeur toutes disponibles sur le site web de Phidgets. [4]

A noter que le nœud compte deux ports pour les capteurs Phidgets, l'un à 3V et l'autre à 5V selon les exigences de chaque capteur.

Tâche 7 : Communication entre deux nœuds

La réalisation d'un réseau de capteurs sans fil complet et bien structuré doit tout d'abord passer par un scénario minimal qui consiste en un réseau de deux capteurs effectuant une communication unicast.

Le premier test qu'on fait c'est d'implémenter les deux codes sur deux zolertia différents l'un joue le rôle de serveur ou sender (alimenté par une batterie) et l'autre de client ou receiver (lié à l'ordinateur pour visualiser la sortie) en utilisant une chaîne de caractère quelconque comme paramètre à envoyer.

Comme on a cité auparavant, le nœud z1 comporte une unité de transmission qui lui permet d'effectuer des communications radio et envoyer des données via le lien radio. Le seul moyen pour exploiter cette caractéristique est l'implémentation des codes adéquats.

Pour réussir cette tâche qui paraissait au début compliquée, nous avons commencé par comprendre le principe des codes qui s'occupent de l'établissement des connexions, comment reconnaître l'adresse du destinataire, les ports de communication, les délais d'envoi et plusieurs autres informations nécessaire au bon fonctionnement du système.

C'est vrai que Contiki est système qui fournit à l'utilisateur une bonne documentation et tant d'exemples de simulations et de codes prêts à être exécutés, mais il n'en demeure pas moins vrai que ces applications ne répondent pas parfaitement aux exigences de tous les projets et doivent être modifié, parfois, complètement bouleversés, pour effectuer nos propres scénarios de communication.

Ainsi, en partant de quelques exemples de communications déjà existants, nous avons pu établir notre scénario et qui répond parfaitement à nos besoins, que ce soit, de point de vue modalités de transmission ou natures des données à transmettre.

Emetteur (Sender) :

Le code source de l'émetteur comporte deux fonctions et évidemment le process Thread.

Fonction receiver :

```

static void
receiver(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{
    printf("Data received on port %d from port %d with length %d\n",
          receiver_port, sender_port, datalen);
}

```

Figure 14 : Fonction receiver

Fonction set-global-address :

```

static void
set_global_address(void)
{
    uip_ipaddr_t ipaddr;
    int i;
    uint8_t state;
    /* Initialize the IPv6 address as below */
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0);
    /* Set the last 64 bits of an IP address based on the MAC address */
    uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
    /* Add to our list addresses */
    uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
    printf("IPv6 addresses: ");
    for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
        state = uip_ds6_if.addr_list[i].state;
        if(uip_ds6_if.addr_list[i].isused &&
           (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
            uip_debug_ipaddr_print(&uip_ds6_if.addr_list[i].ipaddr);
            printf("\n");
        }
    }
}

```

Figure 15 : Fonction set-global-address

Process Thread:

Dans cette partie, nous allons nous intéresser à envoyer la température mesurée par le nœud z1 et l'envoyer à un autre nœud.

```

/*-----*/
PROCESS_THREAD(unicast_sender_process, ev, data)
{
    static struct etimer periodic_timer;
    static struct etimer send_timer;
    uip_ipaddr_t *addr;

    PROCESS_BEGIN();

    servreg_hack_init();

    set_global_address();
    int16_t temp;
    tmp102_init();

    simple_udp_register(&unicast_connection, UDP_PORT,
                       NULL, UDP_PORT, receiver);

    etimer_set(&periodic_timer, SEND_INTERVAL);
    while(1) {

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));
        etimer_reset(&periodic_timer);
        etimer_set(&send_timer, SEND_TIME);

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&send_timer));
        addr = servreg_hack_lookup(SERVICE_ID);
        if(addr != NULL) {
            temp = tmp102_read_temp_x100();

            char buf[20];
            sprintf(buf, "%f", temp);
            printf("Sending temperature reading -> %f via unicast to ", temp);
            uip_debug_ipaddr_print(addr);
            printf("\n");

            simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1, addr);
        } else {
            printf("Service %d not found\n", SERVICE_ID);
        }
    }

    PROCESS_END();
}
/*-----*/

```

Figure 16 : Process Thread

Les principales fonctionnalités apparaissant dans le process Thread sont :

- Etablir l'adresse de l'émetteur
- Initialiser le capteur TMP02
- Fixer l'intervalle d'envoi à SEND_INTERVAL=2 secondes
- Récupérer l'adresse du nœud qui offre un service ayant SERVICE_ID, dans notre cas, c'est le récepteur
- Lire la température
- Envoyer le paquet de données contenant la température à l'adresse déjà récupérée.
- Si l'adresse de l'émetteur est NULL (l'émetteur n'a pas pu établir une adresse pour n'importe quelle raison), le programme affiche le message « Service ID not found ».

Remarque

La fonction *simple_udp_sendto* permet d'envoyer des paquets à une adresse spécifique contrairement à la fonction *simple_udp_send* qui permet d'envoyer des paquets à n'importe quelle adresse disponible.

Récepteur (Receiver) :

Le code source du récepteur comporte deux

Fonction receiver :

```
static void
receiver(struct simple_udp_connection *c,
const uip_ipaddr_t *sender_addr,
uint16_t sender_port,
const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port,
const uint8_t *data,
uint16_t datalen)
{
    char ch[40];

    printf("Data received from ");
    uip_debug_ipaddr_print(sender_addr);
    printf(" on port %d from port %d\n", receiver_port, sender_port);
    if ((receiver_port == UDP_PORT_TEMP) ){

        memcpy(ch, data, VAR_LEN);
        ch[VAR_LEN] = "\0";
        printf("Temperature -> %s\n", ch);
    }
}
```

Figure 17 : Fonction receiver

Cette fonction permet de tester le port de réception des données s'il correspond bien à celui de l'émetteur et de récupérer son adresse. Si c'est vrai, on va copier les données reçues depuis la mémoire vers une variable *ch* de type char puis les afficher.

Fonction set-global-address

Fonction create-rpl-dag

```

static void
create_rpl_dag(uiplib_t *ipaddr)
{
    struct uip_ds6_addr *root_if;
    root_if = uip_ds6_addr_lookup(ipaddr);
    if(root_if != NULL) {
        rpl_dag_t *dag;
        uip_ipaddr_t prefix;
        rpl_set_root(RPL_DEFAULT_INSTANCE, ipaddr);
        dag = rpl_get_any_dag();
        uip_ip6addr(&prefix, 0xaaaa, 0, 0, 0, 0, 0, 0);
        rpl_set_prefix(dag, &prefix, 64);
        PRINTF("created a new RPL dag\n");
    } else {
        PRINTF("failed to create a new RPL DAG\n");
    }
}

```

Figure 18 : Fonction create-rpl-dag

Cette fonction permet de créer un RPL-DAG c'est-à-dire de définir ce nœud récepteur comme un nœud DAG root qui peut recevoir des données depuis les autres nœuds. En effet, le protocole RPL (*Routing Protocol for Low power and Lossy Networks* protocole de routage pour réseaux à basse puissance et avec pertes) est un protocole IPv6 à vecteur de distance qui construit un DAG (de l'anglais *Directed Acyclic Graph*, « graphe orienté acyclique »).

Process Thread :

```

/*-----*/
PROCESS_THREAD(unicast_receiver_process, ev, data)
{
    uip_ipaddr_t *ipaddr;

    PROCESS_BEGIN();

    servreg_hack_init();

    ipaddr = set_global_address();

    create_rpl_dag(ipaddr);

    servreg_hack_register(SERVICE_ID, ipaddr);
    simple_udp_register(&unicast_connection, UDP_PORT_TEMP,
    NULL, UDP_PORT_TEMP, receiver);

    while(1) {
        PROCESS_WAIT_EVENT();
    }
    PROCESS_END();
}
/*-----*/

```

Figure 19 : Process Thread

Les principales fonctionnalités apparaissant dans le process Thread sont :

- Etablir une adresse
- Créer un RPL-DAG
- Enregistrer le service offert par ce nœud
- Enregistrer la connexion UDP et le port du nœud émetteur

Remarque

Servreg-hack-register (*SERVICE_ID,ipaddr*) permet d'enregistrer un service. Enregistrer un service veut dire que les autres nœuds vont être au courant que ce nœud offre un service.

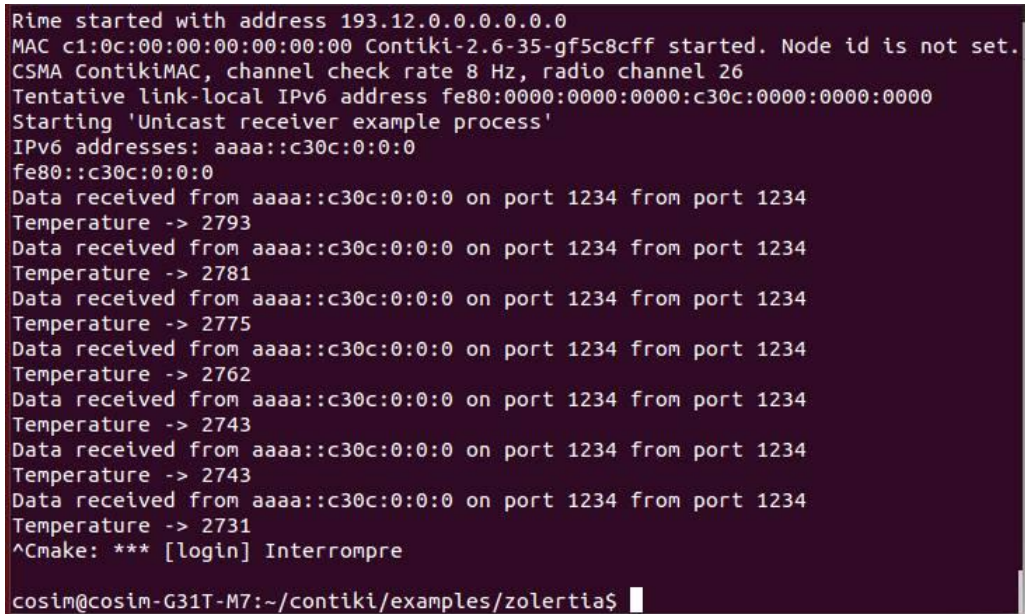
Exécution :

Maintenant que nos codes Sender et Receiver sont prêts, nous commençons comme d'habitude par compiler les programmes puis les implémenter aux nœuds z1.

UDP_PORT_TEMP = 1234

SERVICE_ID=190

On récupère le résultat suivant depuis le nœud Receiver :



```
Rime started with address 193.12.0.0.0.0.0
MAC c1:0c:00:00:00:00:00:00 Contiki-2.6-35-gf5c8cff started. Node id is not set.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0000
Starting 'Unicast receiver example process'
IPv6 addresses: aaaa::c30c:0:0:0
fe80::c30c:0:0:0
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2793
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2781
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2775
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2762
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2743
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2743
Data received from aaaa::c30c:0:0:0 on port 1234 from port 1234
Temperature -> 2731
^Cmake: *** [login] Interrompre
cosim@cosim-G31T-M7:~/contiki/examples/zolertia$
```

Figure 20 : Résultat de l'implémentation des codes Sender et Receiver sur deux noeuds z1

Remarque

La température est multipliée par 100. Ainsi, nous avons vu un exemple de traitement et de transmission de données vers un nœud z1.

Le même principe s'applique pour l'accélération et les capteurs Phidgets.

Prenons le cas de l'accélération : Le code source de Sender est le même sauf dans la partie récupération des données du capteur (sensing).

```

PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&send_timer));
addr = servreg_hack_lookup(SERVICE_ID1);
if(addr != NULL) {
    x = accm_read_axis(X_AXIS);

    etimer_set(&et, ACCM_READ_INTERVAL);
    char buf[20];
    sprintf(buf, "%d", x);
    uip_debug_ipaddr_print(addr);
    printf("\n");
}

```

Figure 21 : Partie du code différente pour l'accélération au niveau du Sender

De même pour le Receiver, le code source ne diffère que dans la fonction receiver, on modifie juste l'affichage pour reconnaître la nature des données reçues selon le port (température ou accélération).

```

static void
receiver(struct simple_udp_connection *c,
const uip_ipaddr_t *sender_addr,
uint16_t sender_port,
const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port,
const uint8_t *data,
uint16_t datalen)
{
    char ch[40];
    printf("Data received from ");
    uip_debug_ipaddr_print(sender_addr);
    printf(" on port %d from port %d\n", receiver_port, sender_port);
    if ((receiver_port == UDP_PORT_ACCEL) ){

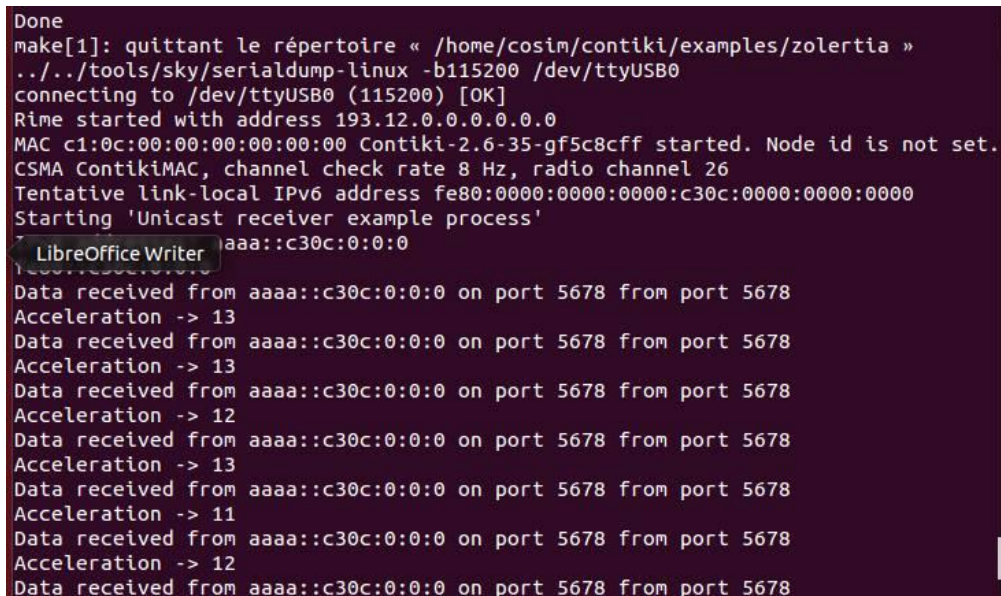
        memcpy(ch, data, VAR_LEN);
        ch[VAR_LEN] = "\0";
        printf("Acceleration -> %s\n", ch);
    }
}
/*-----*/

```

Figure 22 : Fonction receiver

UDP_PORT_ACCEL=5678

On récupère le résultat suivant :



```
Done
make[1]: quittant le répertoire « /home/cosim/contiki/examples/zolertia »
../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Rime started with address 193.12.0.0.0.0
MAC c1:0c:00:00:00:00:00:00 Contiki-2.6-35-gf5c8cff started. Node id is not set.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0000
Starting 'Unicast receiver example process'
aaa::c30c:0:0:0
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 13
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 13
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 12
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 13
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 11
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
Acceleration -> 12
Data received from aaaa::c30c:0:0:0 on port 5678 from port 5678
```

Figure 23 : Résultat de l'implémentation

Tâche 8 : Communication entre 3 zolertia

L'objectif principal du projet consiste à concevoir un réseau de capteurs sans fils à l'aide des nœuds z1 et des capteurs Phidgets. C'est pour ceci, et comme première perception du réseau de capteurs en étoile, nous avons essayé de réaliser le scénario suivant : deux nœuds envoient des données à un nœud central.

On peut nommer les deux nœuds émetteurs Sender1 et Sender2. Le programme C pour les faire fonctionner est exactement le même que le scénario transmission unicast. Evidemment, la différence sera au niveau du récepteur, Receiver, qui doit être capable de recevoir des données depuis deux adresses différentes et surtout, bien les gérer.

Comme on a déjà vu, le récepteur reconnaît l'émetteur depuis deux données principales : l'adresse et le port de communication de l'émetteur.

Ainsi, toute la difficulté réside dans le fait de synchroniser les deux émetteurs pour éviter la perte de données, les interférences, les conflits de transmission...

Dans cette logique-là, nous avons pensé à rendre le récepteur apte à communiquer avec deux émetteurs. Puisque la fonction receiver au niveau du code source du récepteur permet de tester le port de l'émetteur et de récupérer son adresse, nous avons commencé par dupliquer cette fonction, une pour Sender1 et une autre pour Sender2. La seule

différence entre les deux est le port de communication (l'adresse de l'émetteur est automatiquement détectée).

Sender1 : envoyer la température mesurée par z1

Sender2 : envoyer l'accélération mesurée par z1

Au niveau du Receiver :

```
#define UDP_PORT_TEMP 1234
#define UDP_PORT_ACCEL 5678
```

Figure 24 : Ports de communication

```
static void
receiver1(struct simple_udp_connection *c,
const uip_ipaddr_t *sender_addr,
uint16_t sender_port,
const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port,
const uint8_t *data,
uint16_t datalen)
{
char ch1[40];

printf("Data received from ");
uip_debug_ipaddr_print(sender_addr);
printf(" on port %d from port %d\n", receiver_port, sender_port);
if ((receiver_port == UDP_PORT_TEMP)){
memcpy(ch1, data, VAR_LEN);
ch1[VAR_LEN] = "\0";
printf("Temperature -> %s\n", ch1);
}
}
```

Figure 25 : Fonction receiver pour le Sender1

```
static void
receiver2(struct simple_udp_connection *c,
const uip_ipaddr_t *sender_addr,
uint16_t sender_port,
const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port,
const uint8_t *data,
uint16_t datalen)
{
char ch[40];

printf("Data received from ");
uip_debug_ipaddr_print(sender_addr);
printf(" on port %d from port %d\n", receiver_port, sender_port);
if ((receiver_port == UDP_PORT_ACCEL)){
memcpy(ch, data, VAR_LEN);
ch[VAR_LEN] = "\0";
printf("Acceleration -> %s\n", ch);
}
}
```

Figure 26 : Fonction receiver pour le Sender2

Au niveau du process Thread, on conserve, bien évidemment, le numéro de service fourni par le récepteur SERVICE_ID=190 et on appelle la fonction *simple_udp_register* pour les deux émetteurs.

```
simple_udp_register(&unicast_connection, UDP_PORT_TEMP,  
NULL, UDP_PORT_TEMP, receiver1);  
  
simple_udp_register(&unicast_connection, UDP_PORT_ACCEL,  
NULL, UDP_PORT_ACCEL, receiver2);
```

Figure 27 : Appel de la fonction simple-udp-register

Après avoir compilé et exécuté ces trois programmes sur 3 nœuds zolertia, nous avons remarqué que toujours, l'un des deux émetteurs l'emporte sur l'autre et s'accapare de la transmission de ses données. Ainsi, on ne récupère que les données de cet émetteur au niveau du récepteur.

Conclusion

Ce chapitre englobe toutes les tâches qu'on a réalisées tout au long de ce projet.

Conclusion générale

Durant ce projet, nous avons découvert de nouvelles notions concernant l'Internet of Things et plus précisément les réseaux de capteurs sans fils qui devient ces jours-là, un des aspects fondamentaux de M2M et de l'IoT. A l'instar de n'importe quel autre projet, nous avons rencontré plusieurs difficultés, mais, qui étaient vraiment constructives pour avancer et y remédier. Ainsi, ce projet représente un bon commencement dans le domaine de la programmation des capteurs intelligents.

En outre, ce projet nous a donné l'opportunité de rédiger un cahier de charge, de faire l'étude de faisabilité et l'analyse des risques.

Finalement, ce projet était une bonne occasion pour se situer dans le monde professionnel qui nous était jusque-là méconnu et ainsi dépasser les craintes qu'on a envers le monde de l'entreprise.

Webographie

- [1] : http://www.qualimediterranee.fr/imgqualimed/actualite/pdf/veille_captiven_reseaux_de_capteurs.pdf
- [2] : https://fr.wikipedia.org/wiki/R%C3%A9seau_de_capteurs_sans_fil
- [3] : <http://www.wsnmagazine.com/step-by-step-method-of-writing-contiki-programs/>
- [4] : http://www.phidgets.com/docs/1125_User_Guide
- [5] : http://zolertia.sourceforge.net/wiki/index.php/Main_Page

Annexes