TP2 – Programmation avec Contiki
Manipulation d'actionneur et temporisateur (timer)

Important : Un rapport doit être rendu au plus tard 5 jours après le TP. Le compte rendu doit contenir : un rapport avec des captures d'écrans bien expliquées. Le dépôt doit être sur la plateforme «elearning : https://elearning.u-pem.fr/ » et il faut respecter les groupes « apprentis » et « initiaux »

Objectif :

- ☐ Apprendre à développer sous le système d'exploitation *ContikiOS*
- ☐ Programmer le comportement d'un bouton
- ☐ Gestion du temporisateur « timer »

Prérequis :

- ☐ Utilisation de l'émulateur Cooja
- ☐ Le système d'exploitation Contiki

Introduction

Contiki est un système d'exploitation open-source *multitâches orienté évènement*, développé pour des équipements réseaux embarqués (à fortes contraintes de mémoire et de calcul). *Contiki* a deux piles de protocoles différents : ***uIP*** et ***Rime***. ***uIP*** fournit une pile complète TCP/IP afin de supporter la technologie IP et en particulier IPv6/6lowPan. ***Rime*** fournit un support header compressé destiné aux applications qui n'ont besoin que de la couche MAC. Ces deux piles de protocole peuvent être interconnectées. Les données uIP peuvent être transmises sur Rime et vice versa.

Evénements

Le noyau du système *Contiki* est orienté évènement. L'idée d'un tel système est que chaque exécution d'un processus par une application ce n'est qu'une réponse à un évènement.

Les évènements peuvent être classés en trois types :

- **timer events** : sont des évènements basés sur un temporisateur dans le but de générer un évènement après un certain laps de temps. C'est très pratique pour les actions périodiques, ou dans le monde des réseaux comme la synchronisation.
- **external events** : sont des évènements qui proviennent de l'extérieur souvent par des équipements connectés au microcontrôleur par des interfaces d'entrée/Sortie (I/O). Ces équipements lancent des interruptions à chaque fois une actions est détectée (ex. accéléromètre, détecteur de mouvement, etc)
- **internal events** : sont des évènements internes générés par des processus pour communiquer avec d'autres processus (ex. informer le processus de traitement que les données sont prêtes pour le traitement).

Les évènements dits "*posted*" créent une interruption afin de poster un évènement à un processus pendant son exécution. Ces évènements peuvent avoir les informations suivantes :

- **process** : un évènement adressé à un processus. Il peut être destiné à un processus spécifique ou bien à tous les processus enregistrés.
- **event type** : le type d'évènement. Le programmeur peut définir les types d'évènements destinés au processus afin de pouvoir les différencier entre eux (comme le cas de réception de paquet ou de transmission de paquet).
- **data** : certaines données peuvent être générer pour un processus via des évènements.

Ce type d'évènement est le principe de base du système d'exploitation Contiki. Les évènements sont « postés » aux processus. Les processus s'exécutent pendant qu'ils reçoivent les évènements et ils se bloquent pour attendre d'autres évènements lorsqu'il n'y en a plus.

Processus

Le processus est une fonction écrite en C et elle contient des boucles et des appels macro bloquant. Comme Contiki est un système d'exploitation orienté évènements, les processus s'exécutent et ils se bloquent pour attendre des évènements. Plusieurs « macros » sont définies pour différentes possibilités de blocages. L'exemple suivant illustre un processus Contiki :

```
#include "contiki.h"

/* The PROCESS() statement defines the process' name. */
PROCESS(my_example_process, "My example process");

/* The AUTOSTART_PROCESS() statement selects what process(es)
   to start when the module is loaded. */
AUTOSTART_PROCESSES(&my_example_process);

/* The PROCESS_THREAD() contains the code of the process. */
PROCESS_THREAD(my_example_process, ev, data)
{
  /* Do not put code before the PROCESS_BEGIN() statement -
     such code is executed every time the process is invoked. */
  PROCESS_BEGIN();
  /* Initialize stuff here. */
  while(1) {
    PROCESS_WAIT_EVENT();
    /* Do the rest of the stuff here. */
  }
  /* The PROCESS_END() statement must come at the end of the
     PROCESS_THREAD(). */
  PROCESS_END();
}
```

Développer une première application avec Contiki

Dans cette partie, vous allez apprendre à développer un simple programme Contiki. De plus, le processus de développement de la création du projet, l'écriture du programme en langage Contiki, la compilation et l'exécution avec l'émulateur ***MSPsim*** et/ou le ***Mote Tmote Sky***.

Les capteurs Tmote Sky (TelosB) :

Les capteur TelosB ou Tmote Sky (illustré dans la figure en face) sont des modules sans fil à faible consommation d'énergie conçu pour former un réseau de capteur sans fil (WSN : Wireless Sensor Network). Les propriétés de ces capteurs sont :

- Ils intègrent des capteurs d'humidité, de température, et de luminosité.
- Programmable via l'USB

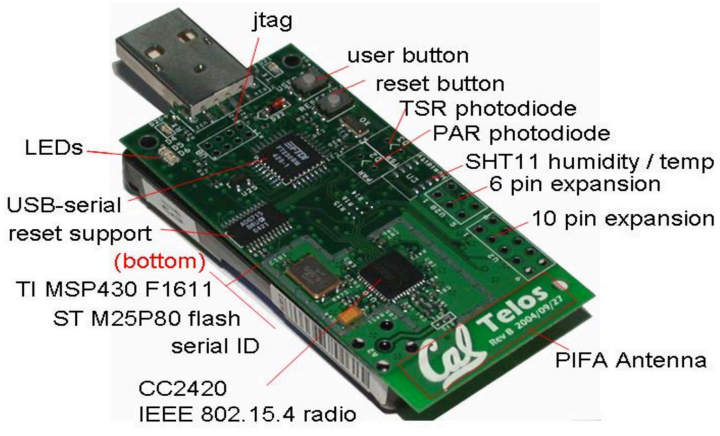

Figure: TelosB Sensor Node

Les étapes à suivre pour développer une application avec Contiki sont :

2. Créer un répertoire pour l'application à développer
3. Créer le fichier Makefile pour la compilation
4. Créer le fichier « **.c* » du programme
5. Exécuter le programme sur le matériel (ex. *Tmote Sky*)

Dans cet exemple, le projet *Contiki* est basé sur code source *C* et le *Makefile*. Le fichier source C contient le programme et le *Makefile* contient les règles de compilation du programme pour générer un exécutable *Contiki*.

1- Création du repertoire du projet

Créer deux répertoires : un pour les projets Contiki et l'autre pour le projet **Hello, Word.**

```
mkdir projects
cd projects
mkdir hello-world-project
cd hello-world-project
```

2- Création du fichier Makefile

Makefile décrit la procédure de compilation du programme. Une fois que le fichier *Makefile* est créé avec votre éditeur préféré (ex. gedit, vi, emacs, …), vous commencez à remplir ce fichier comme suit :

```
CONTIKI=/home/user/contiki
include $(CONTIKI)/Makefile.include
```

Sauvegarder le fichier *Makefile* et commencer à écrire votre premier programme.

3- Créer le fichier du programme

Créer un fichier programme nommé « *hello-world.c* », taper le code ci-dessous :

```
#include "contiki.h"

PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);

PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();
  printf("Hello, world!\n");
  PROCESS_END();
}
```

Commentaire du code :

- La première ligne inclut les en-têtes (header) C du *Contiki*. Cette ligne est nécessaire pour inclure les librairies et les définitions du *Contiki*.
- La deuxième ligne déclare le processus *Contiki*. Ce processus possède une variable nommée (*hello_world_process*) et une chaine de caractère nommée (*Hello world process*). La chaine de caractère est utilisée pour le débogage et comme une commande pour le Shell de Contiki.
- La troisième ligne informe le système Contiki que le programme *hello_world_process* doit se lancer automatiquement lors du démarrage du système.
- La quatrième ligne définie le processus *hello_world_process*. Les arguments ***ev*** et ***data*** dans la macro « *PROCESS_THREAD()* » sont des variables du nombre d'évènement et de données respectivement que le processus peut recevoir.
- Le processus lui-même est définit entre les deux macros : *PROCESS_BEGIN()* et *PROCESS_END()*. Dans ce cas le processus se limite à afficher le message « *hello, world* ».

Sauvegarder le fichier « *hello-world.c* ».

4- Configurer la variable TARGET

Dans la fenêtre terminale, il faut indiquer la plateforme par défaut. Pour ce projet, on peut définir la plateforme « sky » avec cette commande :

```
make TARGET=sky savetarget
```

5- Compilation du projet

A ce stade vous pouvez compiler le projet pour la première fois avec la commande suivante :

```
make hello-world
```

Cette commande entraine à la compilation de l'ensemble du système Contiki.

6- Tester le projet dans MSPsim

Le programme Hello, world peut être testé sans matériel avec l'utilisation du simulateur MSPsim.

```
make hello-world.mspsim
```

La fenêtre « USART1 Port Output » montre la sortie du port série du capteur. Normalement le message « *Hello, world* » est affiché juste après les messages du boot du Contiki.
La fenêtre *SkyGui* montre le capteur *Tmote Sky* avec le Botton « reset » qui nous permet de réinitialiser le capteur. Réinitialiser le capteur à nouveau.
Arrêter la simulation avec *Ctrl+C* ou de fermer simplement la fenêtre.

7- Exécuter le projet sur une plateforme matériel Tmote Sky

- Insérer le capteur Tmote Sky (TelosB) sur le port USB du PC.
- Il faut configurer le VMware Player ou le VirtualBox pour accéder au capteur. Cette procédure peut être effectuée dans le menu de la machine virtuelle. Dans le cas de VMware Player, il faut aller dans (Virtual Machine -> Removable Devices-> future devices mote sky).
- Une fois cette procédure est terminée, le capteur Tmote Sky devient accessible par Instant Contiki.
- Vous pouvez tester cette partie par la commande : ***make sky-motelist*** ou pour afficher les nœuds connectés par la commande **:** ***$tools/sky/motelist-linux***
- Une fois que le capteur est visible via Instant Contiki, il est possible de charger le programme « Hello, World » sur le capteur avec la commande suivante : ***make hello-world.upload***
- Pour exécuter le programme sur les capteurs et voir le résultat de l'exécution, il suffit de taper la commande suivante : ***$make login TARGET=sky***

Remarque :
Si vous avez une erreur de compilation, il faut vérifier si le compilateur ***gcc-msp430*** et la librairie ***msp430-libc*** sont déjà installés sinon ***$sudo apt-get install gcc-msp430 msp430-libc***

Exercice 1 : Commander la couleur des leds via le bouton user

Les capteurs TelosB sont équipés de bouton utilisateur qui permet l'interaction avec le capteur. L'objectif de cet exercice est de programmer ce bouton pour commander la couleur du LED (Vert et Rouge), si le LED est Vert et on appui sur le bouton la couleur du LED passe au Rouge et vice-versa.

Les fonctions du LED et du bouton utilisateur sont dans les fichiers (librairies) suivants : ***/dev/leds.h*** et ***/dev/button-sensor.h***
Afin de réaliser cet exercice vous pouvez suivre les étapes suivantes :

1- **Activation du bouton utilisateur**
Tout d'abord, il faut activer le bouton avant qu'il soit utilisé par l'application. Pour cela, il faut utiliser la fonction **SENSORS_ACTIVATE(button_sensor) ;**
Cette fonction doit être ajoutée dans la section **PROCEE_BEGIN** de l'application.

2- **Déclaration de la section Event**
Dans le but de recevoir des évènements le « **PROCESS_WAIT_EVENT** » doit être intégré dans le code.

3- **Vérification de l'état du bouton**
Le code qui permet de vérifier si le bouton est pressé ou pas est le suivant :

```
if (ev == sensors_event && data == &button_sensor)    {
  printf("Button pressed!");    }
```

Ce code doit être ajouté dans la section **PROCESS_WAIT_EVENT**

4- **Changer la couleur du LED**

Dans le but de changer la couleur du LED du vert vers le rouge et vice-versa, on utilise la fonction **leds_on(LEDS_RED).** Uniquement une seule couleur doit être activé. Les commandes pour contrôler la couleur rouge du LED sont :

```
leds_on(LEDS_RED);
leds_off(LEDS_RED);
leds_toggle(LEDS_RED);
```

Exercice 2 :

L'objectif de cet exercice est la réalisation d'une application de comptage basé sur un *Timer*. Le compteur est incrémenter une fois par second.
Les nœuds TelosB contiennent trois LED qui peuvent être représentés par un nombre de 3 bits. L'idée est d'utiliser les LED pour afficher le modulo 8 (le reste de la division par 8) du compteur comme indiquer dans le tableau ci-dessous :

Counter	Binary	Blue LED	Green LED	Red LED
0	000	Off	Off	Off
1	001	Off	Off	On
2	010	Off	On	Off
3	011	Off	On	On
4	100	On	Off	Off
5	101	On	Off	On
6	110	On	On	Off
7	111	On	On	On

1- Télécharger le fichier « ***comptage-projet.tar.gz*** » sur ma page web.
2- Créer un répertoire « Projet » déposer le fichier « ***comptage-projet.tar.gz*** »
3- Décompresser le fichier avec la commande : ***tar xzvf comptage-projet.tar.gz***
4- Dans le répertoire « ***comptage-projet*** », éditer le fichier ***count.c***
   a. Déclarer un *Timer* et une variable de type entier « *int* » pour le compteur :
      static struct etimer et; static int i;
   b. Initialiser le Timer : ***etimer_set(&et, CLOCK_SECOND) ;***
   c. Déclarer la section event lorsque le Timer s'expire :

```
while(1){
PROCESS_WAIT_EVENT();
if (etimer_expired(&et)){    /* event specific code */    }
}
```

d. Lorsque l'évènement est déclenché :
   - Le contenu de la variable compteur est affiché.
   - Afficher le contenu de la variable du compteur sur les LED selon le tableau précédent.
   - Exemple : il faut utiliser les opérateurs binaires (&, |, ^, ~) pour la manipulation des bits. Si A = 62 et B = 2; A = 0011 1110, B = 0000 0010,
   A&B = 0000 0010, A|B = 0011 1110, A^B = 0011 1100, ~A = 1100 0001.
e. Incrémenter le compteur

5- Réinitialiser le ***Timer***
6- Ajouter une fonction d'initialisation du compteur lorsque le bouton du capteur est pressé. (Afin de simuler le bouton du capteur avec Cooja, il faut utiliser le clic droit sur le nœud et sélectionner "***Click button on Sky X***" (Où X est le ID du nœud)).