TP Programmation d'un kit TI CC2650 à l'aide de Contiki et RIOT

Jérôme Ermont

Février 2024

1 Objectifs

Les objectifs de ce TP sont :

- Prendre en main les fonctionnalités de base de Contiki et RIOT ;
- Tester ces systèmes sur une vraie carte IoT

2 Le matériel

La carte Texas Instrument CC2650STK SensorTag est kit de développement IoT basé sur un micro-contrôleur ARM Cortex-M3. Elle dispose d'une interface Bluetooth Low Energy et IEEE 802.15.4. Elle est constituée :

- d'un thermomètre infrarouge TMP007 qui fournit une température en millième de degré Celsius ;
- d'un capteur de mouvement MPU-9250, composé d'un gyroscope, d'un accéléromètre et d'une boussole ;
- d'un capteur de température et d'humidité HDC1000 ;
- d'un capteur de pression barométrique, de température et d'altitude BMP280 ;
- d'un capteur de lumière ambiente OPT3001.

La figure 1 montre l'emplacement des différents capteurs sur la carte.
Cette carte est compatible avec Contiki-NG et RIOT. Nous allons l'utiliser pour montrer le fonctionnement de ces 2 systèmes d'exploitation utilisés dans le domaine de l'IoT.

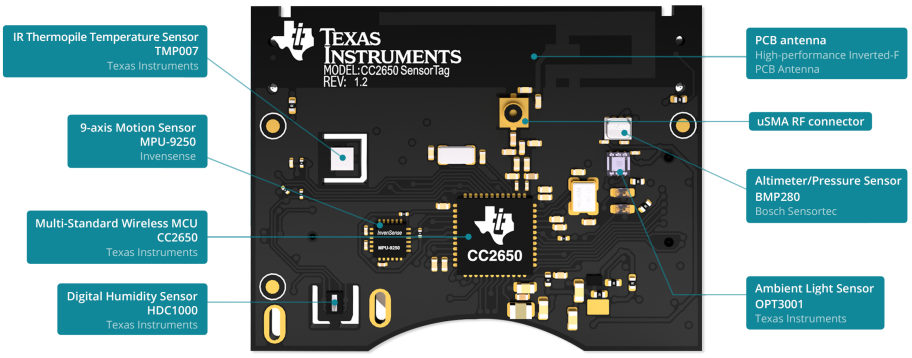

Figure 1 – TI CC2650 SensorTag

3 Initialisation de l'environnement de travail

Les outils (compilateur pour processeur ARM, outil TI pour flasher le circuit) ainsi que Contiki-NG et RIOT sont installés dans le répertoire `/mnt/n7fs/ens/tp_ermont/IoT`. Pour ce faire, exécutez la commande suivante :

```
source /mnt/n7fs/ens/tp_ermont/iot.sh
```

Pour tester la bonne prise en compte de la configuration, tapez :

```
arm-none-eabi-gcc --version
```

Vous devriez visualiser :

```
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10.3-2021.10) 10.3.1 20210824
(release)
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

4 Programmation à l'aide de Contiki-NG

4.1 Un premier test

Pour créer un nouveau projet Contiki-NG, il suffit d'uiliser le script `contiki_create` suivi du nom du projet à créer. Par exemple :

```
contiki_create hello_world
```

Pour cet exemple, le script crée un répertoire nommé `hello_world` composé d'un fichier `Makefile` et d'un fichier source `hello_world.c`. Pour compiler le programme, il suffit d'exécuter la commande `make`.

Une fois la compilation terminée, s'il n'y a pas d'erreur, il est possible de téléverser le programme sur la carte. Le programme à exécuter est Uniflash : `uniflash.sh`
La fenêtre de la figure 2 s'affiche. Il faut choisir le type de circuit : CC2650F128.

Il faut ensuite choisr le debugger/jtag : XDS110 comme le montre la figure 3.

Après avoir appuyer sur le bouton `start`, il faut choisir le fichier à charger da s la fenêtre de la figure 4. Le fichier compilé est situé dans le répertoire du projet dans le sous-répertoire `build/cc26x0-cc13x0/sensortag/cc2650`, par exemple `hello_world.bin`. Puis le chargement s'effecute en appuyant sur le bouton `Load Image`.

Pour visualiser le message, tapez `make login` dans le terminal. Vous devriez voir le message suivant :

```
[INFO: Main      ] Starting Contiki-NG-release/v4.7-43-g999b0d85f
[INFO: Main      ] - Routing: RPL Lite
[INFO: Main      ] - Net: sicslowpan
[INFO: Main      ] - MAC: CSMA
[INFO: Main      ] - 802.15.4 PANID: 0xabcd
[INFO: Main      ] - 802.15.4 Default channel: 26
[INFO: Main      ] Node ID: 52997
[INFO: Main      ] Link-layer address: 0012.4b00.0f0c.cf05
[INFO: Main      ] Tentative link-local IPv6 address: fe80::212:4b00:f0c:cf05
```

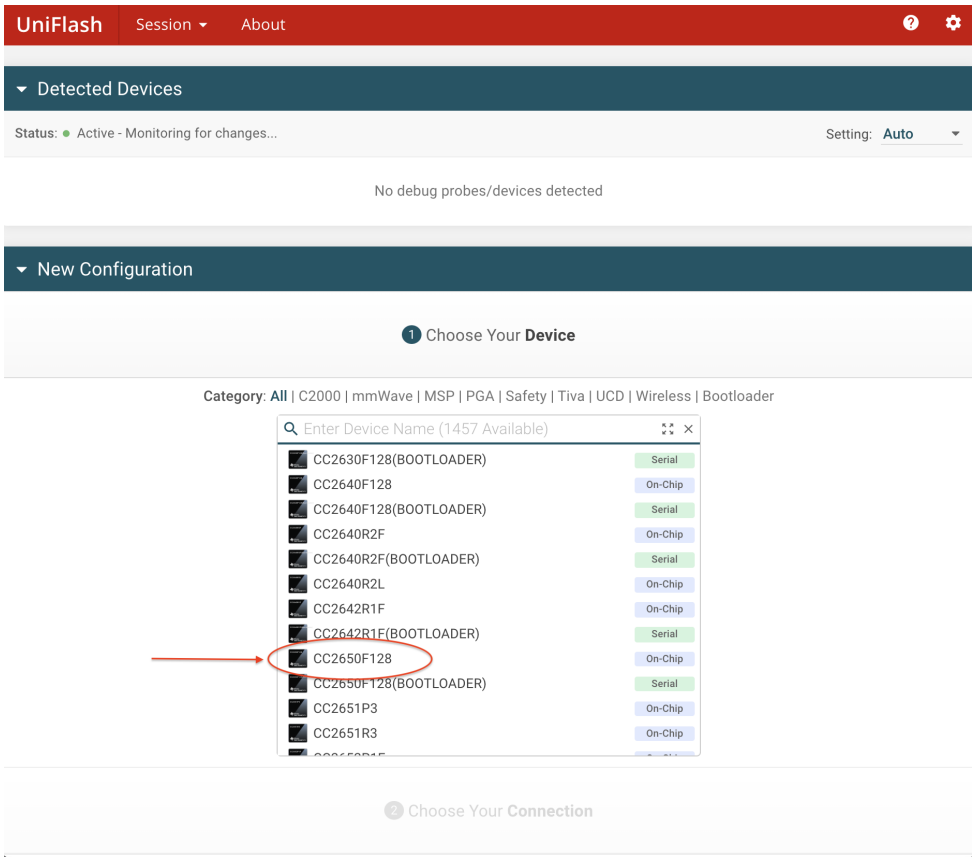

Figure 2 – Uniflash : choisir CC2650F128

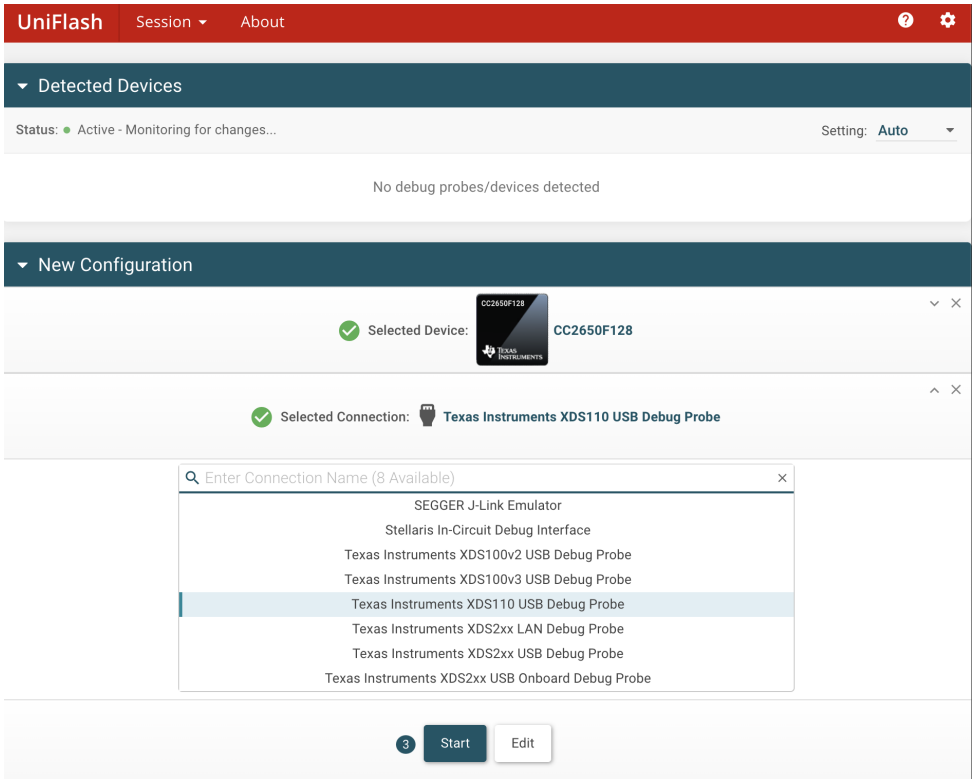

Figure 3 – Uniflash : choisir XDS110

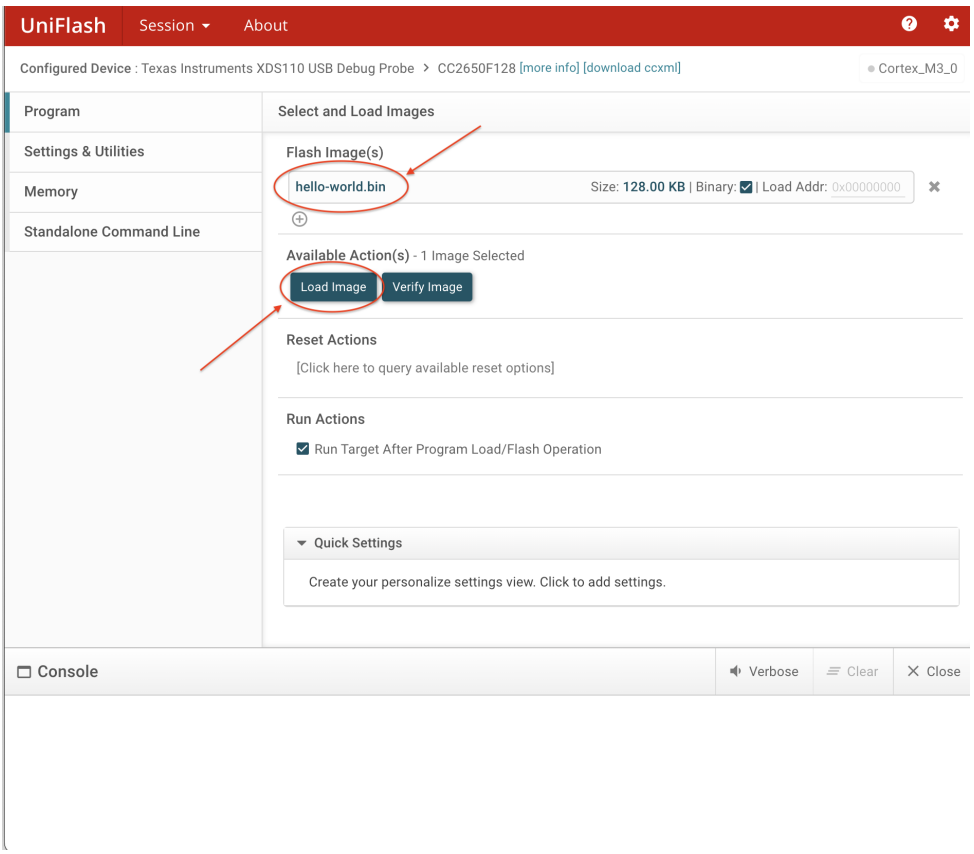

Figure 4 – Uniflash : chargement du fichier

```
[INFO: CC26xx/CC13xx] TI CC2650 SensorTag
[INFO: CC26xx/CC13xx]  RF: Channel 26, PANID 0xABCD
Contiki-NG is running on cc26x0-cc13x0
```

Nous allons utiliser le fichier `hello-world.c` dans la suite.

4.2 Un thread qui lit la température et l'humidité

Nous allons créer le programme qui réalise la lecture des capteurs de température et d'humidité de façon périodique.

Création de processus dans `Contiki-NG`. Pour créer un processus dans `Contiki-NG`, les étapes suivantes sont à suivre :

1. Il faut lui donner un nom.
   La fonction `PROCESS` est là pour ça. Par exemple, le programme `hello-world.c` du listing 1 indique :

   ```
   PROCESS(hello_world_process, "Hello world process");
   ```

   Cela signifie que le processus sera connu du système sous le nom `hello_world_process` et une brève description est donnée.
2. Le lancement du processus se fait par la commande `AUTOSTART_PROCESSES`.
3. Il faut définir le contenu du code du processus :

   ```
   PROCESS_THREAD(hello_world_process, ev, data);
   ```

   Le 1er paramètre est le nom que vous avez donné au processus. Les 2 suivants permettent de gérer les événements que va recevoir le processus, ils sont obligatoires.

4. Enfin un processus commence toujours par la fonction `PROCESS_BEGIN()` et termine toujours par la fonction `PROCESS_END()`.

Gestion du temps. `Contiki-NG` met en œuvre différent type de timers :

— `timer` : il s'agit de lancer un chronomètre qui s'exécute indépendamment du processus. Un timer de ce type n'est donc pas bloquant pour le processus. Un exemple d'utilisation est :
```
timer_set(&mon_timer, 3 * CLOCK_SECOND);
if (timer_expired(&mon_timer)) {
...
```

— `stimer` : ce timer est similaire au précédent. La durée doit être indiquée en secondes.

— `etimer` : ce timer génère un événement qui peut être récupérer par des fonctions telles que `PROCESS_WAIT_EVENT_UNTIL()`. Il est le timer de la programmation événementielle. Par exemple :
```
etimer_set(&mon_timer, 3 * CLOCK_SECOND);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&mon_timer));
...
```
C'est le timer utilisé pour effectuer un affichage périodique dans le listing 1.

— `ctimer` : lance une fonction à expiration du chronomètre. La fonction est placée en paramètre de la fonction `ctimer_set`.
```
void fonction_associee(void *parametre) {
...
}
PROCESS_THREAD() {
ctimer_set(&mon_timer, 3 * CLOCK_SECOND, fonction_associee, parametre);
```

— `rtimer` : Ce timer se déclare comme le précédent. Ce timer sera utilisé lorsque nous souhaitons une action à une date précise.

Pour tout ces timers il faut ajouter l'interface correspondante. Par exemple, pour `etimer` il faudra ajouter :

```
#include <sys/etimer.h>
```

▷ **Exercice 1 : Affichage périodique de** `Hello, World !`
Modifiez le fichier `helloworld.c` *de manière à afficher périodiquement* `Hello, World` *en prenant exemple sur le programme 1.*

Listing 1 – Fichier helloworld.c

```
1  #include "contiki.h"

3  #include <stdio.h> /* For printf() */
   PROCESS(hello_world_process, "Hello␣world␣process");
5  AUTOSTART_PROCESSES(&hello_world_process);
   PROCESS_THREAD(hello_world_process, ev, data)
7  {
     static struct etimer timer;
9
     PROCESS_BEGIN();
11
     /* Setup a periodic timer that expires after 10 seconds. */
13   etimer_set(&timer, CLOCK_SECOND * 10);

15   while(1) {
       printf("Hello,␣world\n");
17
       /* Wait for the periodic timer to expire and then restart the timer
```

```
19      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
        etimer_reset(&timer);
21    }

23    PROCESS_END();
    }
```

▷ **Exercice 2 : Lecture des capteurs**
Editez le programme `helloworld.c`*. Modifiez ce fichier un processus nommé* `helloworld_process` *de façon à ce qu'il lise de façon périodique (toutes les 10s par exemple) les valeurs de la température et de l'humidité fournies par le capteur HDC1000 et les affiche. Pour afficher, utilisez la fonction* `printf("Temp:␣%d.%d\n", temp/100, temp%100);` *où temp est la température lue en centième de degrés.*

> **Remarque :** La documentation du capteur peut être trouvée à l'adresse : `https://github.com/contiki-os/contiki/blob/master/platform/srf06-cc26xx/sensortag/hdc-1000-sensor.h`

4.3 Lancement d'un deuxième thread

La lecture du capteur de pression BMP290 s'effectue de la même manière que précédemment. Nous allons proposer une version multi-programmée en ajoutant ce nouveau capteur.

▷ **Exercice 3 : Version multi-programmée**
Modifiez le programme de manière à avoir 2 processus périodiques (de période différentes, 10s et 4s par exemple) : un qui lit et affiche la température et le taux d'humidité et un deuxième qui lit et affiche la pression.

5 Programmation à l'aide de RIOT

5.1 Un exemple très simple

Pour créer un projet RIOT, utilisez le script `riot_create` suivi du nom de projet.

```
riot_create helloworld
```

Pour compiler le programme, exécutez la commande `make`. Lorsque la compilation est terminée, la commande `make flash` permet de charger le programme sur la carte. Lorsque la programmation est terminée, la commande `make term` affiche les messages. Le résultat ressemble à :

```
This is RIOT! (Version: 2022.04-devel-536-g7c0dd)
You are running RIOT on a(n) cc2650stk board.
This board features a(n) cc26x0_cc13x0 MCU.
```

5.2 Lecture périodique du capteur HDC1000

▷ **Exercice 4 : Lecture du capteur HDC1000**
Modifiez le code du programme `helloworld.c` *de manière à lire les valeurs de température et du taux d'humidité. Utilisez la documentation du capteur HDC1000 à l'adresse* `https://api.riot-os.org/group__drivers__hdc1000.html`

La carte CC2650STK dispose d'un timer. La librairie `ztimer` propose les fonctions qui permettent de contrôler ce timer et contrôler les applications qui sont temporisées. La documentation de cette librairie est disponible via ce lien : `https://doc.riot-os.org/group__sys__ztimer.html`

▷ **Exercice 5 : Affichage périodique de la température et de l'humidité**
Utiliser la librairie `ztimer` *pour afficher en boucle la température et le taux d'humidité. Le programme attendra 2s entre 2 affichages.*

5.3 Programmation avec plusieurs threads

La création de thread en RIOT se fait à l'aide de le fonction :

```
kernel_pid_t thread_create
    ( char *                stack,      // la pile du thread
      int                   stacksize,  // taille de la pile
      uint8_t               priority,   // priorite
      int                   flags,      // options
      thread_task_func_t    task_func,  // fonction a executer
                                        // par le thread
      void *                arg,        // arguments de la fonction
      const char *          name        // nom du thread
        )
```

Pour chaque thread, il faut définir une pile qui permettra de stocker la dynamique de l'exécution.

- Une pile se définit comme un tableau d'octets dont la taille peut être une valeur par défaut, une taille mininmale, égale à la taille de la pile de l'appelant, ... Les constantes sont définies dans le fichier `thread.h` (`https://doc.riot-os.org/group__core__thread.html`)
- La plus petite valeur de priorité est possédée par la tâche de priorité la plus forte. Il est ici aussi possible d'utiliser les constantes de RIOT pour avoir la priorité minimale, la priorité de la tâche appelant, ...
- les options permettent de définir comment le thread doit se lancer.
- la fonction exécutée par le thread aura le format : `(void *) function (void *arg)`.

▷ **Exercice 6 : Création d'un premier thread**
Créer un premier thread qui réalise la lecture périodique des valeurs du capteurs HDC1000. Exécuter le code sur la carte.

▷ **Exercice 7 : Création d'un deuxième thread**
Créer un deuxième thread qui lit périodique la luminosité ambiante toutes les 5s. Observer l'exécution du code.