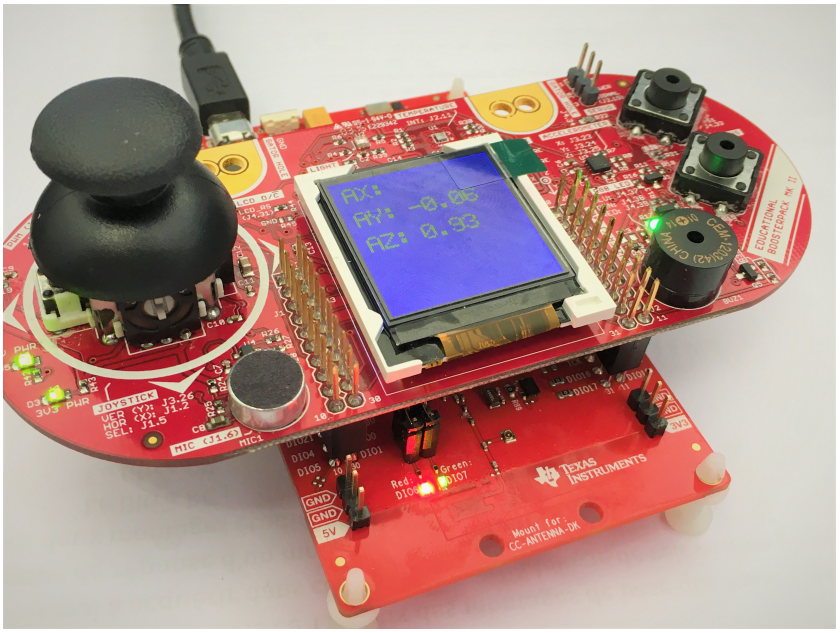
Master ROC :
Real-Time Operating Systems - IoT - TP2



1 Introduction

Le TP2 consiste en développer un capteur de température connecté avec une alarme paramétrable. Les données seront affichés sur un écran LCD et envoyés par BLE.

Le système d'exploitation en temps réel utilisé est *TI-RTOS*, ceci est fourni par *Texas Instruments* également que le hardware prise en main. Le System-on-Chip (SoC) CC2652R. Il est conseillé d'avoir fait le TP1 avant de commencer celle-ci.

1.1 Architecture du capteur connecté

Un firmware basé en RTOS es composé par différents tâches, le capteur connecté à développer est base sur 4 tâches.

La figure montre le schéma de tâches à implémenter.

1. La première tâche : Task I2C s'occupe de l'échantillonnage de capteur de température. Ensuite cette tâche va envoyer les données aux tâches : Task LCD et Task BLE.
2. La deuxième tâche : Task ADC s'occupe de l'échantillonnage du joystick en utilisant l'ADC. Ensuite cette tâche va envoyer les données aux tâches : Task LCD et Task BLE.
3. La troisième tâche : Task LCD va afficher les données reçues sur l'écran LCD et fera une alerte avec le buzzer quand le niveau de température est supérieur au niveau fixé par le joystick.
4. La dernière tâche : Task BLE va envoyer les données par une liaison Bluetooth.

Finalement les données envoyées par BLE seront visualisées avec un application iOS ou Android de type BLE Scanner.

La section 2 montre les logiciels à installer pour le développement du capteur. Ces logiciels sont déjà installés sur les ordinateurs de la salle donc c'est possible de passer directement a la section 3.

2 Installation et prise en main de l'environnement de développement

L'environnement de développement se compose de 2 logiciels couplés :

1. Code Composer Studio (CCS) : CCS permet de gérer des projets utilisant des microcontrôleurs de Texas Instruments.
2. SimpleLink : SimpleLink est une plateforme de microcontrôleurs qui dispose d'un portefeuille d'unités MCU (system-on-chip) Arm® câblées et sans fil dans un seul environnement de développement de logiciel à partir d'un code C.

Ces deux logiciels permettent donc de programmer un microcontrôleurs de Texas Instruments à l'aide d'un code en C.

2.1 Installation de Code Composer Studio

Cette partie peut être optionnelle si les logiciels sont déjà installés sur votre machine. Elle est toutefois présentée afin que vous puissiez réaliser l'installation sur vos ordinateurs (les logiciels sont gratuits) pour travailler sur vos projets.

1. Vous pouvez télécharger les fichiers d'installation des 2 logiciels dans leurs dernières versions à l'aide des liens suivantes :
   Code Composer Studio
   SimpleLink CC13x2-26x2 SDK
2. Installer si ça n'est pas déjà fait CCS, en laissant les options par défaut, sauf quand il demande sur la sélection de composants. Ajouter *SimpleLink CC13xx and CC26xx Wireless MCUs*.
   Redémarrer si besoin.
3. Installer ensuite SimpleLink CC13x2-26x2 SDK si ça n'est pas déjà fait, en laissant également les options par défaut. SimpleLink sera intégré automatiquement à CCS, sans action de votre part.
4. Au lancement de CCS, un message peut vous demander le *workspace* à utiliser. laissez l'adresse par défaut.
5. Branchez a présent le kit de développement *LAUNCHXL-CC26X2R1* sur le PC. Il doit être reconnu par Windows. En cas de problème demandez qu professeur.

3 Échantillonage des capteurs par le bus I2C

Pour ce projet nous allons partir, également que le TP1, du projet simple peripheral et nous allons créer les taches de la même façon.

La première tache a développer est en charge de l'échantillonnage du capteur de température intégré dans le kit *BOOSTXL-EDUMKII*. Pour cela il faudra chercher le datasheet du capteur et regarder les registres à lire pour obtenir la température. Ensuite pour le code du protocole I2C il faudra chercher l'exemple dans le dossier : $examples \rightarrow rtos \rightarrow CC26X2R1\_LAUNCHXL \rightarrow drivers \rightarrow i2ctmp \rightarrow tirtos \rightarrow ccs$ et copier les lignes importants.

N'oubliez pas d'ajouter le bus I2C dans l'interface sysconfig et sélectionner les pins comme nous avons fait dans le TP1 pour les drivers.

Après avoir vérifié les données en mode debug nous allons passer à la tache suivante.

4 Échantillonage du joystick avec l'ADC

Ceci nous l'avons déja fait dans le TP1 donc vous n'avez qu'à refaire cette partie du code.

Configurez la valeur obtenue par l'ADC de façon de pouvoir paramétrer une alerte de température entre 0 et 33 degrés.

5 Affichage de données et génération de l'alerte

Ceci nous l'avons déjà fait dans le TP1 donc vous n'avez qu'à refaire cette partie du code. Ensuite pour l'alerte vous allez utiliser un timer à 1 *[KHz]* et le connecter dans le pin du buzzer. Quand la température et supérieur à celle configurée par le joystick on active cet alerte.

6 Envoie de données par BLE

Finalement nous allons envoyer les données par BLE. Cependant, dans ce cas nous avons deux taches asynchrones qui doivent envoyer des données. Donc vous allez réfléchir quelle est la meilleur et plus simple manière d'envoyer ces données et l'implémenter. Pour l'implémentions prenez l'exemple du TP1.

Il faudra également envoyer par BLE un message l'alerte quand elle est générée.

Pensez à créer différents services avec les caractéristiques nécessaires.

Exemple de création d'une tache

```
/*
* TacheLCD.c
*
*  Created on: 12 nov. 2022
*      Author: Sebastian MARZETTI
*/

#include <stdbool.h>

#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <math.h>

#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/BIOS.h>

#include <ti_drivers_config.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/SPI.h>

#include <TacheLCD/TacheLCD.h>
#include <driverlib/ssi.h>


#define     TACHELCD_TASK_PRIORITY      1
#define     TACHELCD_TASK_STACK_SIZE    1024

Task_Struct TacheLCD;
uint8_t     TacheLCDStack[TACHELCD_TASK_STACK_SIZE];

Semaphore_Struct semTacheLCDStruct;
Semaphore_Handle semTacheLCDHandle;

extern void TacheLCD_init(void);


static void TacheLCD_taskFxn(UArg a0, UArg a1)
{
        for(;;)
        {
        }
}
```

```
void TacheLCD_CreateTask(void){
        Semaphore_Params semParams;
        Task_Params taskParams;
        /* Configuration de la tache*/
        Task_Params_init(&taskParams);
        taskParams.stack = TacheLCDStack;
        taskParams.stackSize = TACHELCD_TASK_STACK_SIZE;
        taskParams.priority = TACHELCD_TASK_PRIORITY;
        /* Creation de la tache*/
        Task_construct(&TacheLCD, TacheLCD_taskFxn,
        &taskParams, NULL);

        /* Construire un objet semaphore
        pour etre utilise comme outil de
        verrouillage, comptage initial 0 */
        Semaphore_Params_init(&semParams);
        Semaphore_construct(&semTacheLCDStruct,
        0, &semParams);
        /* Obtenir la gestion de l'instance */
        semTacheLCDHandle =
        Semaphore_handle(&semTacheLCDStruct);
}
```