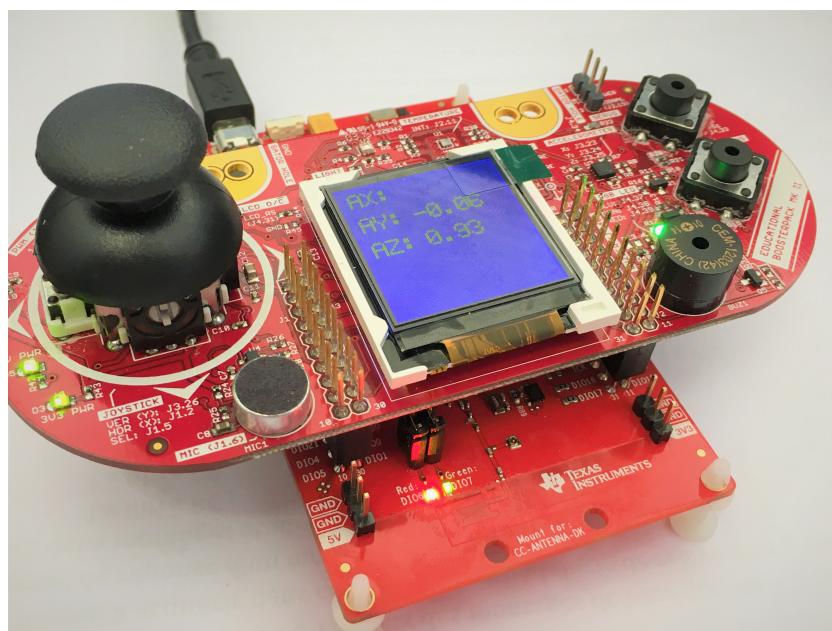


Master ROC :

Real-Time Operating Systems - IoT



1 Introduction

Ce TP présente une initiation aux systèmes d'exploitation en temps réel et à l'IoT. Un capteur (accéléromètre) connecté sera développé. 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.

1.1 Architecture du capteur connecté

Un firmware basé en RTOS es composé par différents tâches, le capteur connecté à développer est base sur 3 tâches.

La figure 1 montre le schéma de tâches à implémenter.

1. La première tâche : Task ADC s'occupe de l'échantillonnage de l'accéléromètre analogique. Ensuite cette tâche va envoyer les données aux tâches : Task LCD et Task BLE.
2. La deuxième tâche : Task LCD va afficher les données reçues sur l'écran LCD.
3. La troisième tâche : Task BLE va envoyer les données par une liaison Bluetooth.

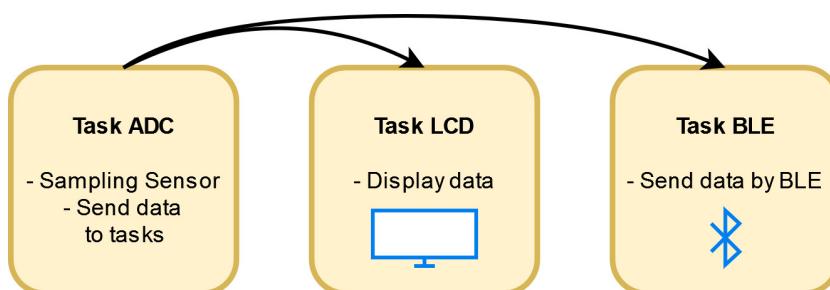


FIGURE 1 – Schéma de tâches du capteur

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 à 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 à présent le kit de développement *LAUNCHXL-CC26X2R1* sur le PC. Il doit être reconnu par Windows. En cas de problème demandez qu'professeur.

3 Premier programme

Cette partie utilise le kit de développement *LAUNCHXL-CC26X2R1*, qui est décrit un détail sur le site internet de chez Texas Instruments *LAUNCHXL-CC26X2R1*. C'est donc cette carte qu'il faudra programmer, elle intègre un CC2652R1. Le programmeur est intégré aussi dans la carte donc il faudra juste connecter le kit de développement par USB.

Nous allons utiliser aussi un deuxième kit de développement appelé *BOOSTXL-EDUMKII* décrit un détail aussi sur le site de chez Texas Instruments *BOOSTXL-EDUMKII*.

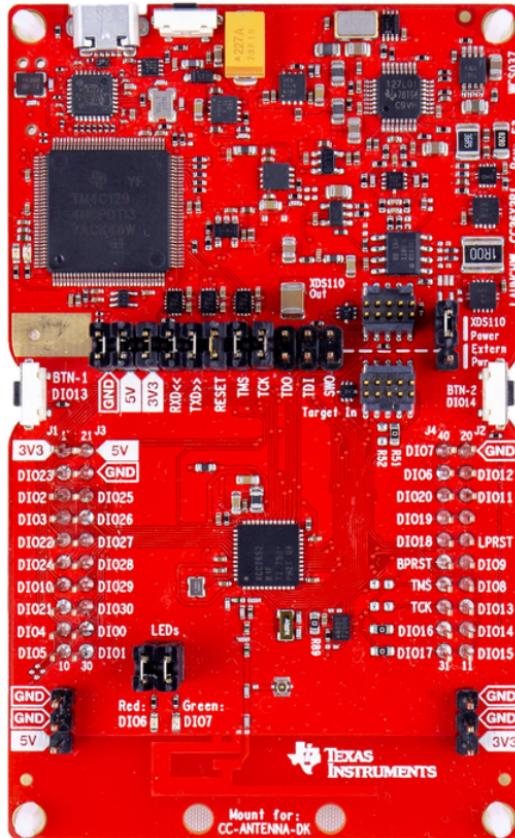


FIGURE 2 – LAUNCHXL-CC26X2R1

Le SoC (System on Chip) CC26x2R1 utilisé est décrit en détail sur le site internet de chez Texas Instruments : CC2652R.

Vous trouverez sur le site du constructeur toutes les informations nécessaires au fonctionnement de chacun des systèmes du SoC avec des exemples de code permettant de les utiliser.

3.1 Cration du projet

Nous allons utiliser un exemple ou toute la gestion du BLE soit djà faite. Pour cela, lancer CCS, puis allez dans :

Project → Import CCS Projects → Browse

Cherchez le dossier *simplelink_cc13x2_26x2_sdk_X_XX_XX_XX*, par daut il est install sur *C:/ti*. puis allez sur le projet «simple peripheral» et slectionnez le dossier *CCS* qui est dedans le dossier *tirtos* :

```
C:/ti/simplelink_cc13x2_26x2_sdk_5_20_00_52/examples/rtos/
CC26X2R1_LAUNCHXL/ble5stack/simple_peripheral/tirtos/ccs
```

Le projet devrait tre sur l'explorateur de projets comme il est montr dans la figure 3.

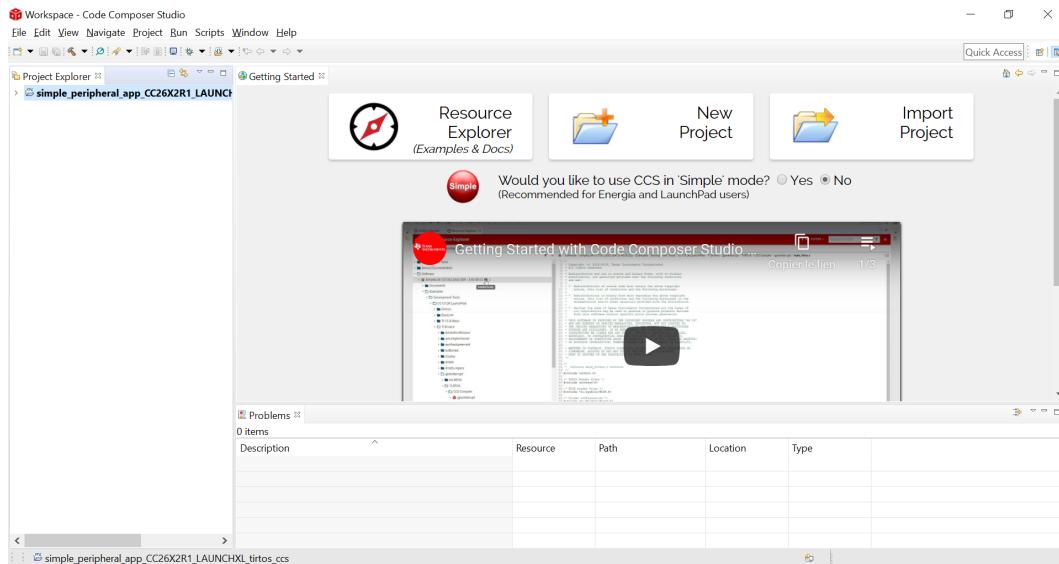


FIGURE 3 – Projet «simple peripheral» importé

Si votre projet a été bien importé vous pouvez le compiler avec l’icône représentant un marteau ou avec :

Project → Build Project

A l’issue de la compilation il ne devrait pas y avoir des erreurs donc flashez le SoC avec (icône à droite pour flash et icône à gauche pour debug). Au cas qu’il y ait des erreurs demandez au professeur.

Une fois que le kit de développement est flashé, regardez avec une tablet ou smartphone en utilisant l’App *BLE Scanner* ou équivalent si l’identifiant BLE du projet est affiché. Vous pouvez vérifier le nom de l’exemple dans le fichier : *simple_peripheral.syscfg*. Dans la figure 4 on observe le fichier syscfg et le nom du capteur (*Simple Peripheral*).

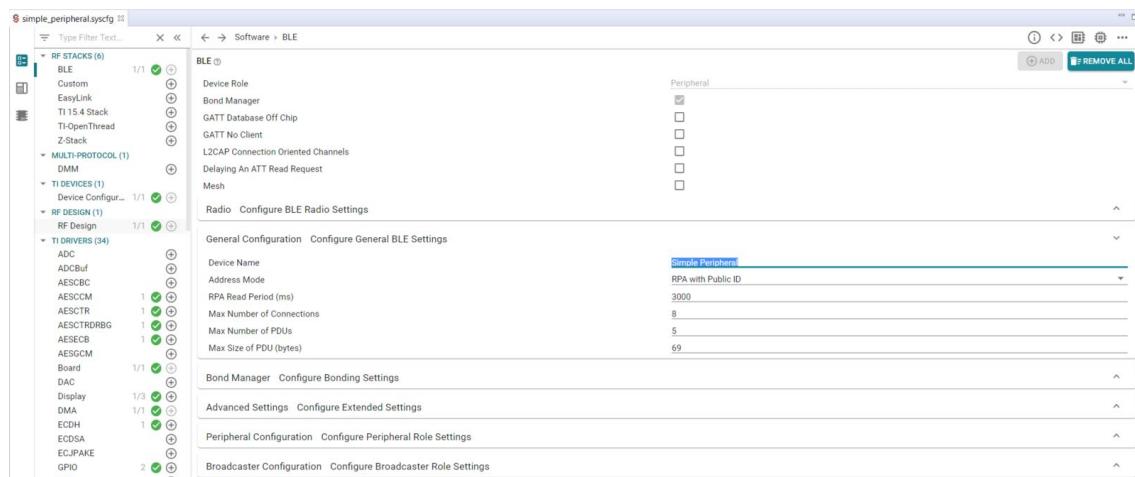


FIGURE 4 – Fichier syscfg, BLE device name

3.1.1 Modification du nom du capteur

Choisissez un nom pour votre capteur et modifiez-le comme montre les figures suivantes.

1. Modification du *Device Name*, la figure 5 montre où il faut modifier le nom du dispositif. Ceci sera le *Complete Local Name*, dans ce cas il est *Sensor Group 1*, cependant vous pouvez choisir celui que vous préférez.

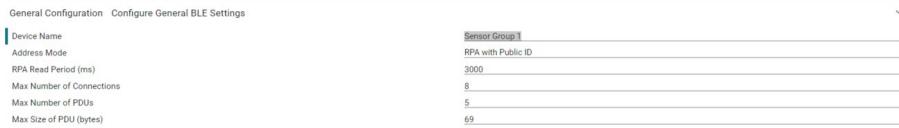


FIGURE 5 – BLE device name modification

2. Dans la section *Broadcaster Configuration* nous allons modifier les suivants paramètres :
 - (a) Modification du *Advertisement Data 1* dans la section *Advertisement Set 1*, la figure 6 montre la configuration nécessaire pour ce paramètre. Le *shortened Local Name* doit avoir 2 caractères.



FIGURE 6 – Advertisement Data 1 modification

- (b) Modification du *Scan Response Data 1* dans la section *Advertisement Set 1*, la figure 7 montre la configuration nécessaire pour ce paramètre. Le *Complete Local Name* doit être pareil à celui que nous avons configuré pour le Device Name.

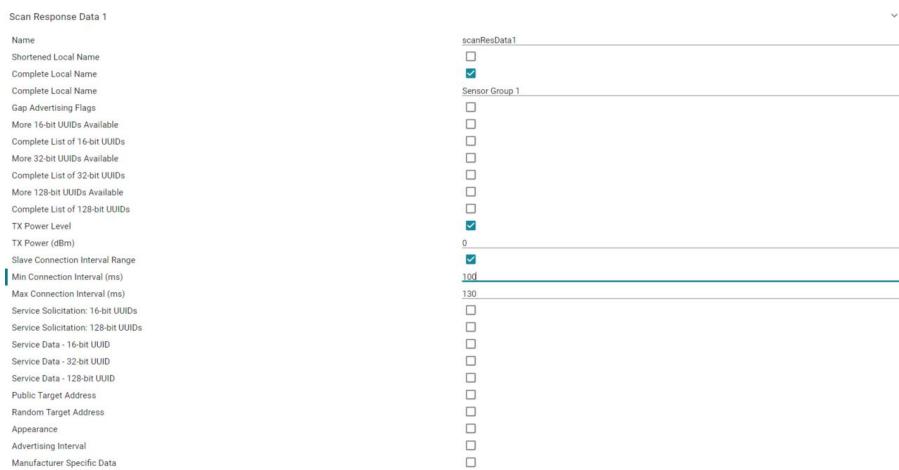


FIGURE 7 – Scan Response Data 1 modification

- (c) Modification du *Advertisement Set 2*, la figure 8 montre la configuration nécessaire pour ce paramètre.

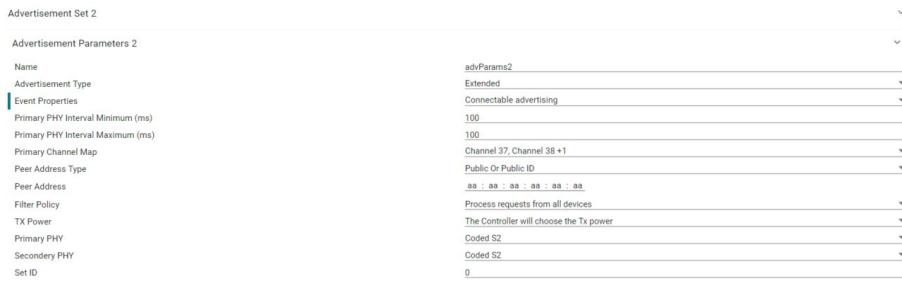


FIGURE 8 – Advertisement Set 2 modification

- (d) Modification du *Advertisement Data 2*, la figure 9 montre la configuration nécessaire pour ce paramètre. Le *shortened Local Name* doit avoir 2 caractères.

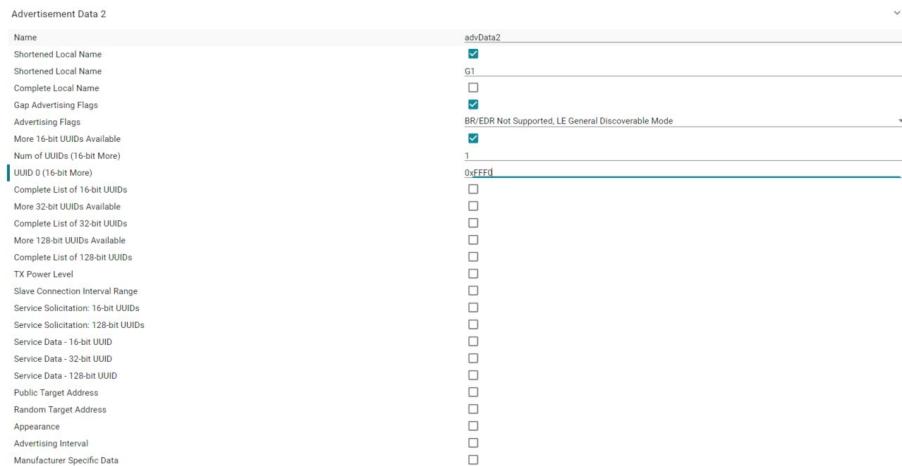


FIGURE 9 – Advertisement Data 2 modification

- (e) Modification du *Scan Response Data 2*, la figure 10 montre la configuration nécessaire pour ce paramètre. Le *Complete Local Name* doit être pareil à celui que nous avons configuré pour le Device Name.

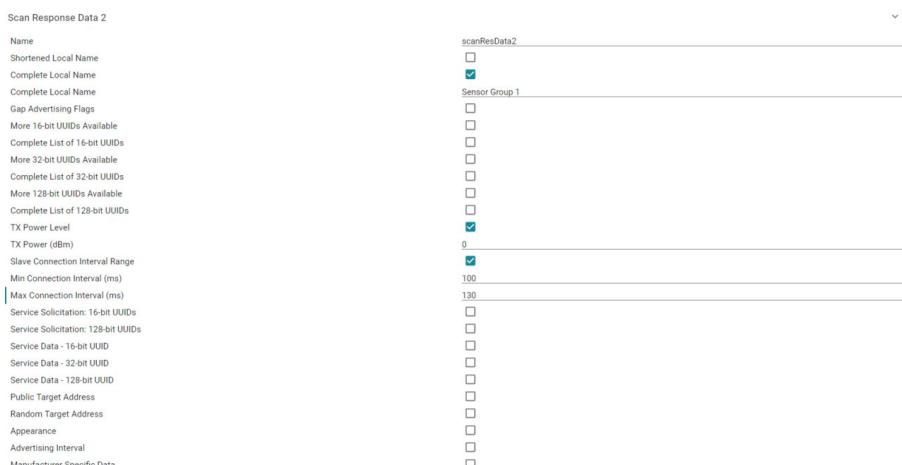


FIGURE 10 – Scan Response Data 2 modification

3.2 Crédation de la tâche ADC

Après avoir vérifié le l'advertising du système avec le nom que nous avons choisi, nous allons créer une tâche pour l'échantillonnage de l'accéléromètre intégré sur le kit BOOSTXL-EDUMKII montré dans la figure .

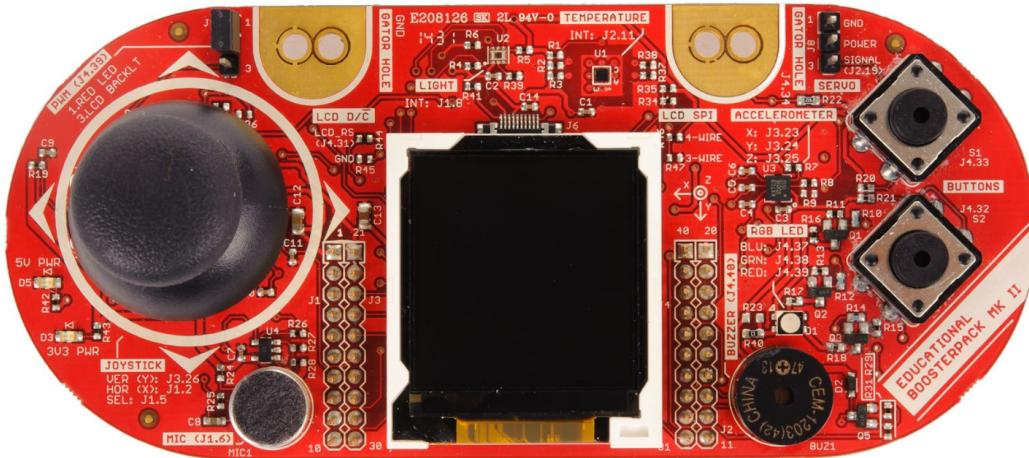


FIGURE 11 – BOOSTXL-EDUMKII

- Créer un nouveau dossier appelé «TacheADC» sur le projet :

Faites click droit sur le projet et ensuite :

New → Folder

Folder Name : TacheADC → Finish

- Créer un fichier .h et un fichier .c appellés TacheADC sur le dossier qu'on vient de créer :

Faites click droit sur le dossier «TacheADC» et puis :

New → Header File

Header file : TacheADC.h → Finish

New → Source File

Source file : TacheADC.c → Finish

- En commençant par le fichier source qu'on vient de créer, inclure les suivants lignes :

```
#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 <TacheADC/TacheADC.h>
```

4. Nous allons configurer la tâche avec un priorité de 1 (Maximum 5) et la taille du stack de 1024, ensuite il faut déclarer la structure de la tâche et son stack et finalement nous allons déclarer un semaphore lequel va permettre de bloquer la tâche quand il est nécessaire :

```
#define TACHEADC_TASK_PRIORITY 1
#define TACHEADC_TASK_STACK_SIZE 1024
Task_Struct TacheADC;
uint8_t TacheADCStack[TACHEADC_TASK_STACK_SIZE];
Semaphore_Struct semTacheADCStruct;
Semaphore_Handle semTacheADCHandle;
```

5. Créer la fonction qui effectuera la tâche. Elle sera la fonction main de notre tâche avec son propre boucle infini :

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

6. Finalement nous allons créer la fonction chargé de la creation de la tâche. Elle doit être appelée dans le main principal (dossier Startup) pour qu'elle soit effectivement créée :

```
void TacheADC_CreateTask(void){
    Semaphore_Params semParams;
    Task_Params taskParams;
    /* Configuration de la tache*/
    Task_Params_init(&taskParams);
    taskParams.stack = TacheADCStack;
    taskParams.stackSize = TACHEADC_TASK_STACK_SIZE;
    taskParams.priority = TACHEADC_TASK_PRIORITY;
    /* Creation de la tache*/
    Task_construct(&TacheADC, TacheADC_taskFxn,
                  &taskParams, NULL);

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

On vient de créer une tâche avec un semaphore, cette méthode est générale donc il nous permet de créer n'importe quel type de tache et l'adapter à nos besoins. Dans le cas où le semaphore n'est pas nécessaire, on peut enlever tout ce qui le concerne dans la méthode.

Cette tâche est dédiée à l'échantillonnage de l'accéléromètre, donc il faut initialiser le driver ADC. Pour cela nous allons ouvrir l'interface syscfg (Fig. 12).

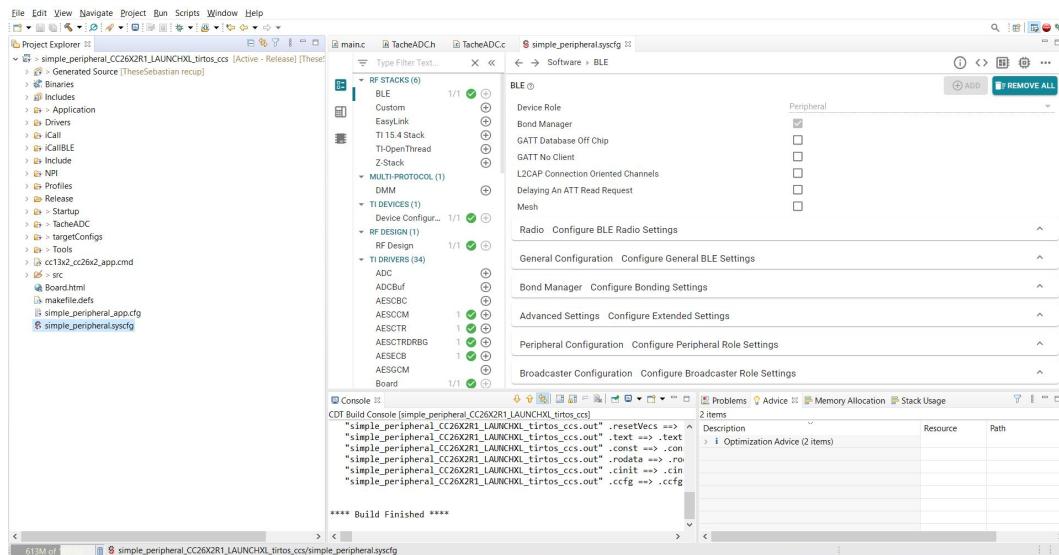


FIGURE 12 – Interface graphique syscfg

L'accéléromètre utilisé a 3 axes donc il faut ajouter 3 channels de l'ADC (Fig. 13).

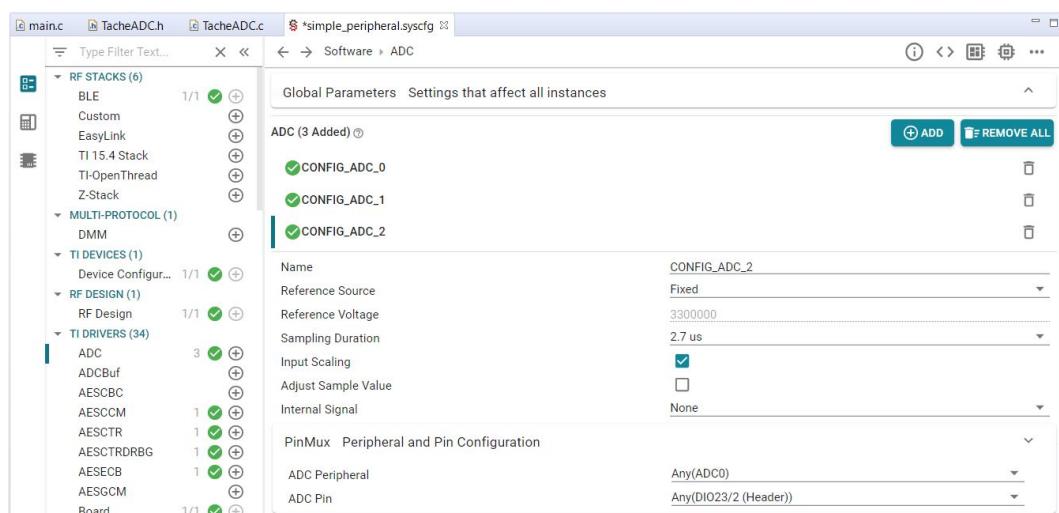


FIGURE 13 – Interface graphique syscfg : ADC Peripheral

Ne pas oublier de configurer les pins d'entrée de chaque un des channels de l'ADC. Pour cela vérifier dans la carte où arrivent les sorties de l'accéléromètre analogique

Ensuite, il faut ajouter au fichier *TacheADC.c* les références suivantes qui correspondent au driver de l'ADC, ainsi que le callback du timer virtuel (myClock) que l'on va créer après :

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

```
#include <ti/drivers/ADC.h>

#define ADC_SAMPLE_COUNT (10)

uint16_t adcValue0;
uint32_t adcValue0MicroVolt;
uint16_t adcValue1[ADC_SAMPLE_COUNT];
uint32_t adcValue1MicroVolt[ADC_SAMPLE_COUNT];

static Clock_Struct myClock;

extern void TacheADC_init(void);
void Sampling(uint_least8_t Board_ADC_Number);

void myClockSwiFxn(uintptr_t arg0)
{
    Semaphore_post(semTacheADCHandle);
}

uint16_t i;
ADC_Handle adc;
ADC_Params params;
int_fast16_t res;
```

Notre fonction *TacheADC_taskFxn* doit consister des suivantes lignes :

```
static void TacheADC_taskFxn(UArg a0, UArg a1)
{
    TacheADC_init();
    for(;;)
    {
        //Le semaphore est poste par le timer myClock
        Semaphore_pend(semTacheADCHandle, BIOS_WAIT_FOREVER);
        Sampling(CONFIG_ADC_0);
        Sampling(CONFIG_ADC_1);
        Sampling(CONFIG_ADC_2);
    }
}
```

Chaque fois que le timer arrive à 100 millisecondes, il va poster le semaphore et on fera l'échantillonnage des 3 axes de l'accéléromètre. Donc il faudra créer le timer, ceci est fait à continuation :

```
extern void TacheADC_init(void){
    GPIO_init();
    ADC_init();
    ADC_Params_init(&params);
    Clock_Params clockParams;
    Clock_Params_init(&clockParams);
    clockParams.period = 100 * (1000/Clock_tickPeriod), //100ms
    Clock_construct(&myClock, myClockSwiFxn,
```

```

    0, // Initial delay before first timeout
    &clockParams);
    Clock_start(Clock_handle(&myClock)); //Timer start
}

```

Finalement il faut ajouter la fonction Sampling, où on fera l'échantillonnage de chaque axe de l'accéléromètre. L'ADC ne peut pas fonctionner avec plusieurs canaux en simultanée donc à toute conversion il faudra ouvrir et fermer le canal avec *ADC_open* et *ADC_close*

```

void Sampling(uint_least8_t Board_ADC_Number){
    adc = ADC_open(Board_ADC_Number, &params);
    if (adc == NULL){
        while (1);
    }
    for (i = 0; i < ADC_SAMPLE_COUNT; i++){
        res = ADC_convert(adc, &adcValue1[i]);
        if (res == ADC_STATUS_SUCCESS){
            adcValue1MicroVolt[i] =
                ADC_convertRawToMicroVolts(adc,
                                            adcValue1[i]);
        }
    }
    ADC_close(adc);
}

```

3.3 Initialisation du système d'exploitation

On a fini notre tâche. Maintenant il faut la créer dans l'initialisation du système RTOS. Cela se passe dans le fichier main.c donc inclure le .h de la tâche et création de la fonction (Figs. 14 et 15).

```

main.c ✘ TacheADC.h TacheADC.c simple_peripheral.syscfg ti_drivers_config.h
77
78 #include <TacheADC/TacheADC.h>
79 /*****

```

FIGURE 14 – fichier .h

```

main.c ✘ TacheADC.h TacheADC.c simple_peripheral.syscfg ti_drivers_config.h
155
156     SimplePeripheral_createTask();
157     TacheADC_CreateTask();
158     /* enable interrupts and start SYS/BIOS */
159     BIOS_start();
160
161     return 0;
162 }

```

FIGURE 15 – Tâche ADC

Une fois qu'on a fini notre tâche, il faut que le système d'exploitation sache qu'elle existe et qu'il doit la créer. Pour ça nous allons au dossier *Startup*, fichier

main. C'est là où le système est initialisé et toutes les tâches sont créées. Donc déjà il faut inclure le fichier header contenant notre fonction de création de la tâche et l'ajouter juste avant de la création de la tâche *simple peripheral*, (Figs. 14 et 15).

3.4 Debug

Pour tester nous allons debugger. Mettre un breakpoint et lancer le debug comme montre la Fig. 16.

```

File Edit View Navigate Project Run Scripts Window Help
Project Explorer [ simple_peripheral_CC26X2R1_LAUNCHXL_tirtos [Active - Release] [These]
  > Generated Source [TheseSebastian recup]
  > Binaries
  > Includes
  > Application
  > Drivers
    > NV
    > iCall
    > iCallBLE
    > Include
    > NPI
    > Profiles
    > Release
    > Startup
    > TacheADC
    > targetConfigs
    > Tools
    > ccl3x2_cc26x2_app.cmd
  > src
    > Board.html
    > makefile.defs
    > simple_peripheral_app.cfg
    > simple_peripheral.syscfg

```

```

81   Clock_start(Clock_Handle(&myClock)); //Timer start
82 }
83
84 void Sampling(uint_least8_t Board_ADC_Number){
85   adc = ADC_open(Board_ADC_Number, &params);
86   if (adc == NULL){
87     while (1);
88   }
89   for (i = 0; i < ADC_SAMPLE_COUNT; i++){
90     res = ADC_convert(adc, &adcValue1[i]);
91     if (res == ADC_STATUS_SUCCESS){
92       adcValue1MicroVolt[i] = ADC_convertRawToMicroVolts(adc, adcValue1[i]);
93     }
94   }
95   ADC_close(adc);
TacheADC.c, line 95 ($\$L22 + 0xE) [H/W BP]
96 }
97
98
99 void TacheADC_CreateTask(void){
100   Semaphore_Params semParams;
101   Task_Params taskParams;
102   /* Configuration de la tache*/
103   Task_Params_init(&taskParams);
104   taskParams.stack = TacheADCStack;
105   taskParams.stackSize = TACHEADC_TASK_STACK_SIZE;
106   taskParams.priority = TACHEADC_TASK_PRIORITY;
107   /* Creation de la tache*/
108   Task_construct(&TacheADC, TacheADC_taskFxn,

```

FIGURE 16 – Lancer le debug

Mettre un breakpoint comme montre la Fig. 17 et vérifier le fonctionnement de l'adc pour les 3 axes.

Expression	Type	Value	Address
adcValue1MicroVolt	unsigned int[10]	[1646496, 1648592, 1647552, 1648592, ..., 0x200260C, 0x200260C]	
↳ [0]	unsigned int	1646496	0x200260C
↳ [1]	unsigned int	1645592	0x2002610
↳ [2]	unsigned int	1647552	0x2002614
↳ [3]	unsigned int	1645592	0x2002618
↳ [4]	unsigned int	1645592	0x200261C

```

File Edit View Project Tools Run Scripts Window Help
Debug 33
Texas Instruments XDS110 USA Debug Probe/Cortex,M4,0 Suspended - HW Breakpoint
  Sampling() at TacheADC.c:95 0x001366
  TacheADC_TaskFn() at TacheADC.c:66 0x00161E
  Swap_restore() at C:/Github/TheseSebastian/workspace/vacaine2022/simple_peripheral_CC26X2R1_LAUNCHXL_tirt
  ADCCCC26XX.convertToMicroVolts() at ADCCCC26XX.c:400 0x0011999
  TI_syslib_TaskSupportProxy_swap_EI() at C:/Github/TheseSebastian/workspace/vacaine2022/simple_periphera
  ATYYYYYYXV convertRawToMicroVolts() at ATM_Confir <inlin> at ATYYYYYYXV-0x0011999

```

```

main.c in TacheADC.c TacheADC.c simple_peripheral.syscfg ti_drivers_config.h
84 void Sampling(uint_least8_t Board_ADC_Number{
85   adc = ADC_open(Board_ADC_Number, &params);
86   if (adc == NULL){
87     while (1);
88   }
89   for (i = 0; i < ADC_SAMPLE_COUNT; i++){
90     res = ADC_convert(adc, &adcValue1[i]);
91     if (res == ADC_STATUS_SUCCESS){
92       adcValue1MicroVolt[i] = ADC_convertRawToMicroVolts(adc, adcValue1[i]);
93     }
94   }
95   ADC_close(adc);
96 }
97
98
99 void TacheADC_CreateTask(void){
100   Semaphore_Params semParams;
101   Task_Params taskParams;
102   /* Configuration de la tache*/
103   Task_Params_init(&taskParams);
104

```

FIGURE 17 – Lancer le debug

Si tout fonctionne correctement nous pouvons passer à la tâche LCD.

3.5 Création de la tâche LCD

Après avoir vérifié les mesures de l'ADC sur chaque voie de l'accéléromètre nous allons afficher ces données sur l'écran LCD intégré dans le kit BOOSTXL-EDUMKII montré dans la figure 11.

Une fois que le convertisseur analogique numérique fini ses conversions il va poster le semaphore de la tâche LCD qui lui donnera feux vert pour afficher les données. Donc nous allons **créer une nouvelle tache pour contrôler l'écran LCD**, pour ça on va **utiliser la méthode expliquée en 3.2**.

L'écran est contrôlée par le protocole SPI, donc il faudra initialiser les périphériques concernés.

1. Créer la tâche SPI (dossier et fichiers correspondants) avec un priorité de 1 et un stack size de 1024
2. Ajouter un périphérique SPI avec l'interface syscfg comme montre la Fig. 18.

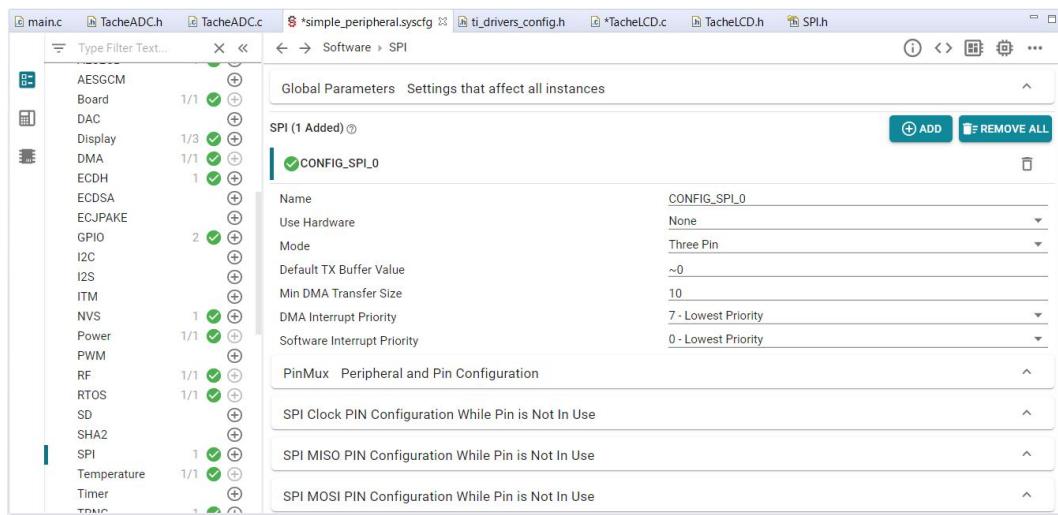


FIGURE 18 – Add peripheral SPI

3. Configurer les pins du bus SPI comme montre la Fig. 19.

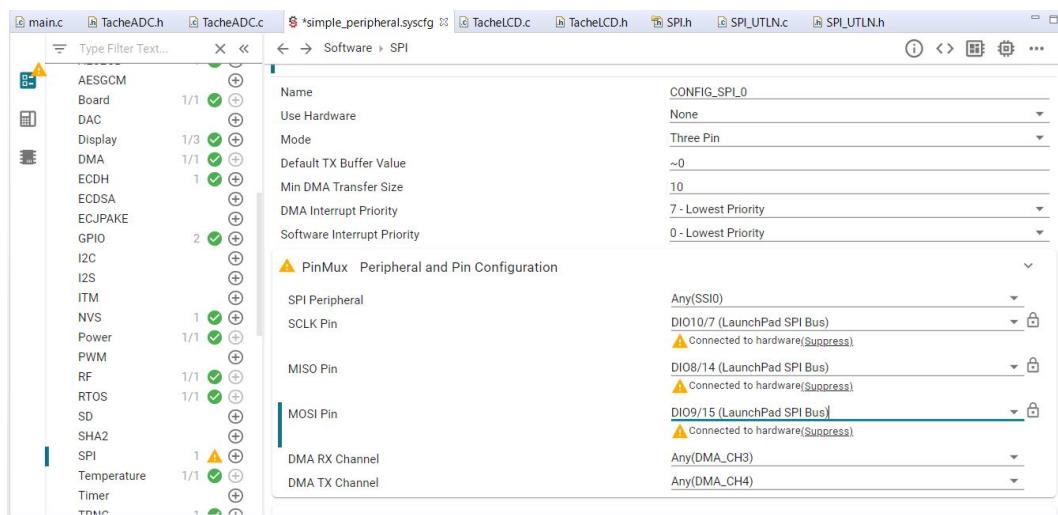


FIGURE 19 – Pins bus SPI

4. Finalement pour le LCD que nous allons utiliser il nous faut un pin concernant le chip select de la liaison SPI et un autre pour chip reset. Pour cela, nous

allons au fichier de configuration (simple_peripheral.syscfg), d'abord il configurer les deux GPIO CONFIG_GPIO_BTN1 et CONFIG_GPIO_BTN2 en mode *Use Hardware* : *None* comme montre la figure 20.

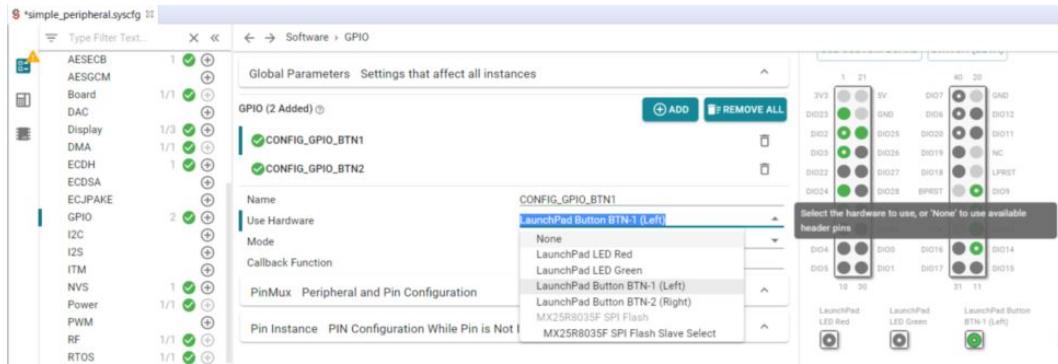


FIGURE 20 – Configuration des boutons du kit de développement

Ensuite nous ajoutons deux GPIO. Le premier il faut le nommer SPI_LCD_CS, le configurer en sortie et l'assigner au pin DIO13 et le deuxième SPI_LCD_RS, le configurer en sortie aussi et l'assigner au pin DIO17 (Fig. 16). **Les deux sont configurés en sortie (OUTPUT)** (Fig. 21).

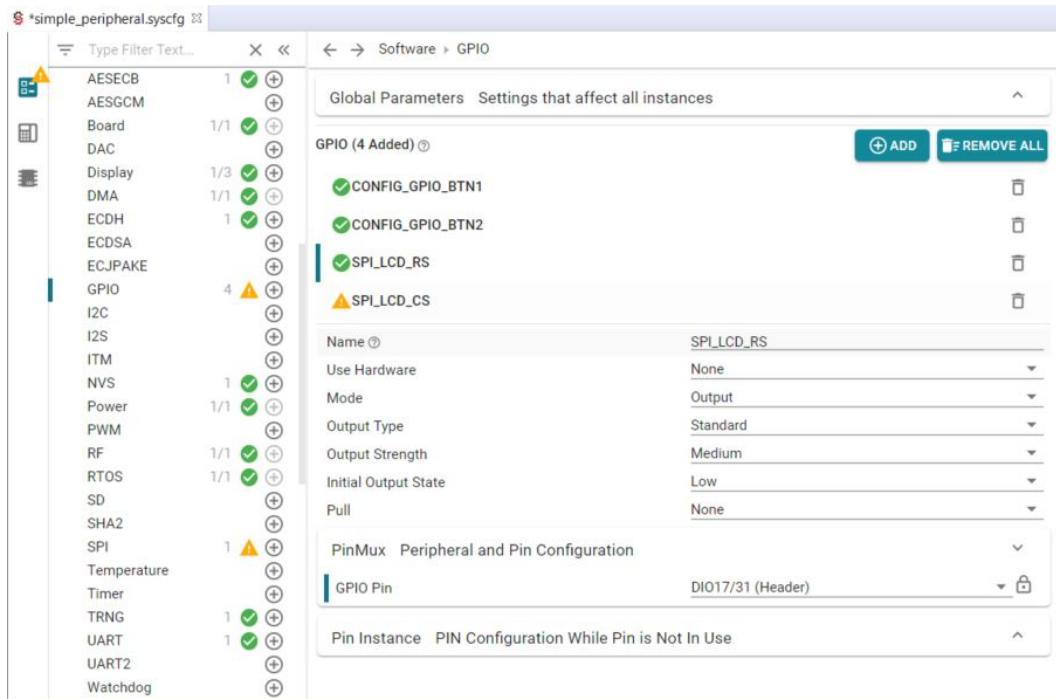


FIGURE 21 – Pins SPI chip select (CS) et reset (RS)

5. Ajouter les fichiers *LCD_LAUNCHPAD* et *SPI_UTLN* au dossier *TacheLCD*.
6. Ajouter les lignes suivantes :

```
#include "TacheLCD/LCD_LAUNCHPAD/OLED_Display.h"
#include <TacheLCD/SPI_UTLN/SPI_UTLN.h>
#include <TacheLCD/LCD_LAUNCHPAD/LCD_LAUNCHPAD.h>
extern void TacheLCD_init(void);
```

```

void LCD_Init(void);
void floatToString(char* ax, float AX);
void floatToString1d(char* ax, float AX);
void intToString(char* ax, float AX);
float Ax,Ay,Az;

void afficherDonnees(float accx, float accy, float accz){
    Ax = accx;
    Ay = accy;
    Az = accz;
    Semaphore_post(semTacheLCDHandle);
}

void LCD_Init(void){
    //Initialize the LCD controller
    Initialize_LCD();

    //Fill display with a given RGB value
    Fill_LCD(0xFF,0x00,0x00); //RGB
    Task_sleep(500000/Clock_tickPeriod); // Delay 100ms

    char DataLCD[] = "UTLN";

    OLEDText22( 40, 17, DataLCD, SIZE_TWO, 0x00, 0xFF, 0xFF );
    Task_sleep(1000000/Clock_tickPeriod); // Delay 1s
}

void TacheLCD_init(void){
    SPI_UTLN_Init();
    LCD_Init();
}

void floatToString1d(char* ax, float AX){
    char convert[6];
    //char* ax = " ";
    if(AX<0){
        char moins[6] = "-";
        strcat(ax,moins);
        AX = AX*-1.0f;
    }
    char point[6] = ".";
    long Axlong = (long)AX;
    long Axdeclong = (long)((AX - (float)Axlong)*10);
    ltoa(Axlong,convert,10);
    strcat(ax,convert);
    strcat(ax,point);
    ltoa(Axdeclong,convert,10);
}

```

```

        strcat(ax,convert);
    }

void intToString(char* ax, float AX){
    char convert[6];
    //char* ax = " ";
    if(AX<0){
        char moins[6] = "-";
        strcat(ax,moins);
        AX = AX*-1.0f;
    }
    long Axlong = (long)AX;
    ltoa(Axlong,convert,10);
    strcat(ax,convert);
}

```

7. Ensuite il faut coder la tâche LCD. La base est montrée dans la Fig. 22.

The screenshot shows the Code Composer Studio interface with the following details:

- Project Explorer:** Shows the project structure under "simple_peripheral_CC26X2R1_LAUNCHXL_tirtos_ccs". It includes generated source files, binaries, includes, application drivers, and peripheral components like TacheADC, TacheLCD, SPI, and UTLN.
- Code Editor:** Displays the source code for the LCD task. The code includes function prototypes for `TacheLCD_init` and `TacheLCD_taskFxn`, and a main loop that converts an ADC value to a string and displays it on the LCD.
- Console:** Shows build logs for the project, indicating successful initialization of the Cortex-M4 core and board reset.
- Problems:** Shows 0 errors, 1 warning, and 0 others.
- Resource Usage:** Shows memory allocation and stack usage details.

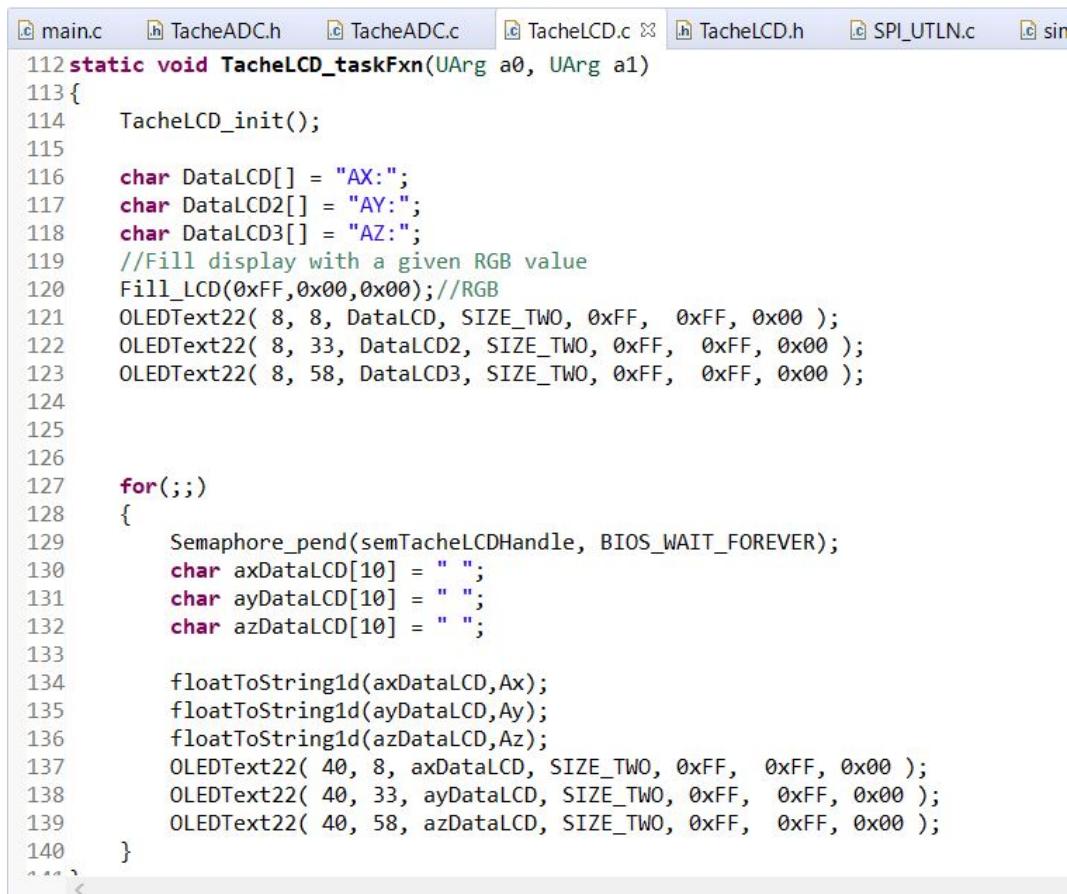
FIGURE 22 – Code de base pour la tâche LCD

1. La communication entre les deux taches se fera avec la fonction `afficherDonnees` donc cette fonction sera appelé dans la Tâche ADC. Ne pas oublier de l'inclure dans le fichier .h.
2. Cette fonction reçoit des float et la sortie de l'ADC est un entier donc il faut trouver l'information dans le manuel du composant (lien ici KXTC9-2050) pour faire la conversion de int à float.
Sensitiviy = 660 mV/G
3. Compléter les lignes de code manquants pour afficher les données reçues de la tâche ADC. Il faudra convertir float à string donc pensez à utiliser les fonctions fournies avant

4. Ne pas oublier d'appeler la création de la tâche LCD dans le main principal

5. Faire les modifications nécessaires pour convertir les valeurs en [G] ($1 \text{ [G]} = 9.81 \text{ [m/s}^2\text{]}$) et les envoyer en format float à la tâche LCD

Une des possibles solutions pour la tâche LCD est montrée dans la Fig. 23



```

112 static void TacheLCD_taskFxn(UArg a0, UArg a1)
113 {
114     TacheLCD_init();
115
116     char DataLCD[] = "AX:";
117     char DataLCD2[] = "AY:";
118     char DataLCD3[] = "AZ:";
119     //Fill display with a given RGB value
120     Fill_LCD(0xFF,0x00,0x00); //RGB
121     OLEDText22( 8, 8, DataLCD, SIZE_TWO, 0xFF, 0xFF, 0x00 );
122     OLEDText22( 8, 33, DataLCD2, SIZE_TWO, 0xFF, 0xFF, 0x00 );
123     OLEDText22( 8, 58, DataLCD3, SIZE_TWO, 0xFF, 0xFF, 0x00 );
124
125
126
127     for(;;)
128     {
129         Semaphore_pend(semTacheLCDHandle, BIOS_WAIT_FOREVER);
130         char axDataLCD[10] = " ";
131         char ayDataLCD[10] = " ";
132         char azDataLCD[10] = " ";
133
134         floatToString1d(axDataLCD,Ax);
135         floatToString1d(ayDataLCD,Ay);
136         floatToString1d(azDataLCD,Az);
137         OLEDText22( 40, 8, axDataLCD, SIZE_TWO, 0xFF, 0xFF, 0x00 );
138         OLEDText22( 40, 33, ayDataLCD, SIZE_TWO, 0xFF, 0xFF, 0x00 );
139         OLEDText22( 40, 58, azDataLCD, SIZE_TWO, 0xFF, 0xFF, 0x00 );
140     }
141 }
```

FIGURE 23 – Code de pour la tâche LCD

3.6 Crédation d'un service BLE

Une fois qu'on aura affiché les données sur l'écran et on aura vérifié les mesures en [G] de l'accéléromètre on va envoyer aussi les données par BLE. Donc pour ça on va d'abord créer des services BLE avec ses caractéristiques.

Nous allons utiliser un générateur du code pour créer le service : Générateur du code, dans la section *Example service generator*.

Le nom du service va être *Accelerometre* (on va éviter les accents lors qu'on fait du code) et l'UUID du service : *0xBA55* d'un taille de 16 bits. Après on va ajouter un caractère (avec le bouton *Add Characteristic*) dont le nom sera *AccelerometreMesures*, l'UUID *0x2BAD* d'un taille de 128 bits et la taille *Value len*

égal à 20. Finalement on va sélectionner les propriétés *GATT_PROP_READ*, *GATT_PROP_WRITE* et *GATT_PROP_NOTIFY* et les permissions *GATT_PERMIT_READ* et *GATT_PERMIT_WRITE*. Les configurations doivent être comme celles de la figure 24.

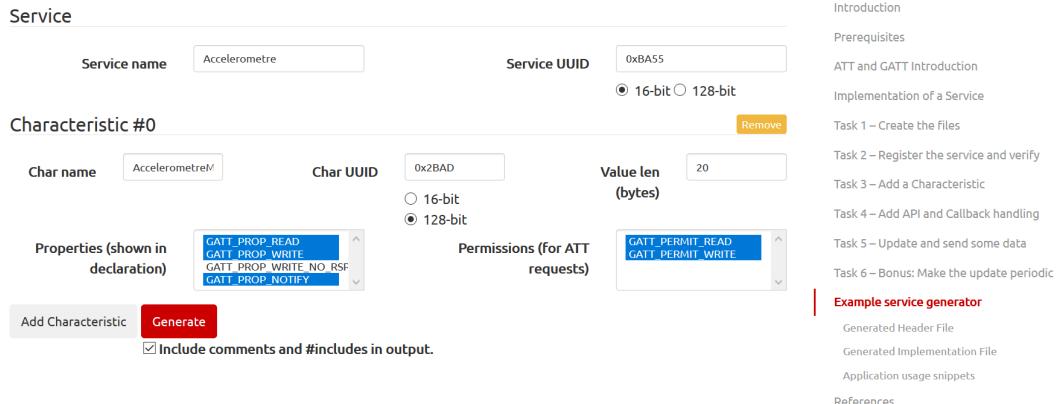


FIGURE 24 – Création du profile

On les génère avec *Generate* et ensuite on récupere les fichiers .h et .c pour les ajouter dans le dossier *Profiles* comme montre la fig. 25.

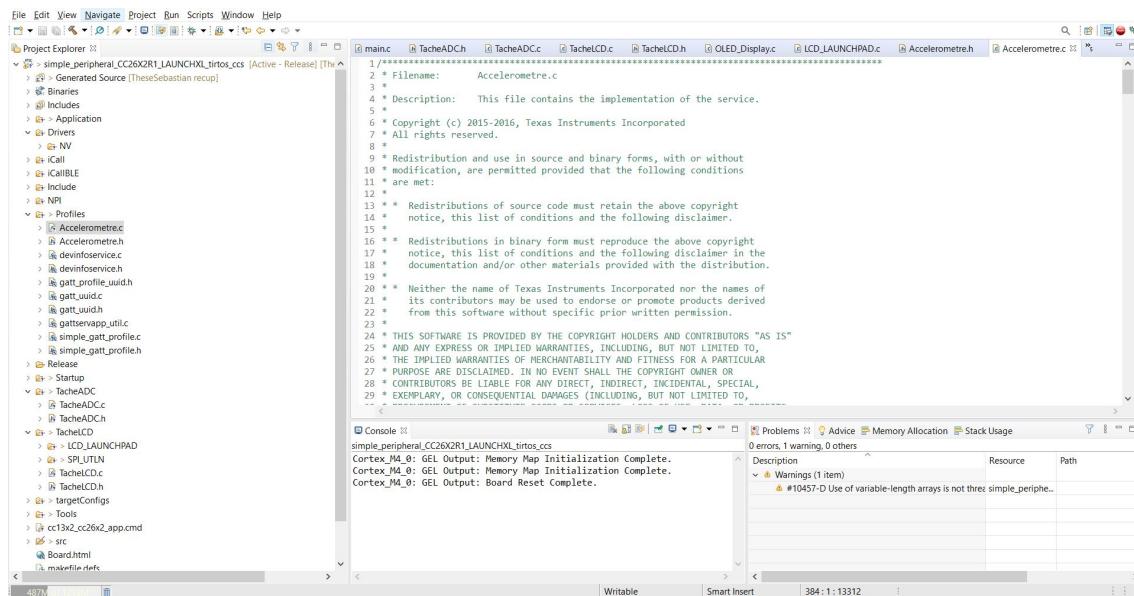


FIGURE 25 – Profiles generated

Il y a aussi une section appelée *Application-snippets*, qui sera utilisée dans 3.7. On va nommer les fichiers source et header : *Accelerometre.c* et *Accelerometre.h* comme montre la fig. 25 sans accents.

Au moment de la compilation on aura des erreurs dus aux problèmes de la version, donc pour ça on va inclure les suivants fichiers dans le fichier source :

```
#include "icall_ble_api.h"
#include "icall.h"
```

Et on va remplacer :

```
INVALID_HANDLE par LINKDB_HANDLE_INVALID
```

3.7 Modification de la tâche Simple Peripheral

La tâche chargée de la gestion du BLE est *Simple Peripheral* donc c'est ici où on va ajouter le profile qu'on vient de créer. Pour ça on va ajouter les suivantes lignes. On va inclure le fichier header du profil qu'on vient de faire :

```
#include "Profiles/Accelerometre.h"
```

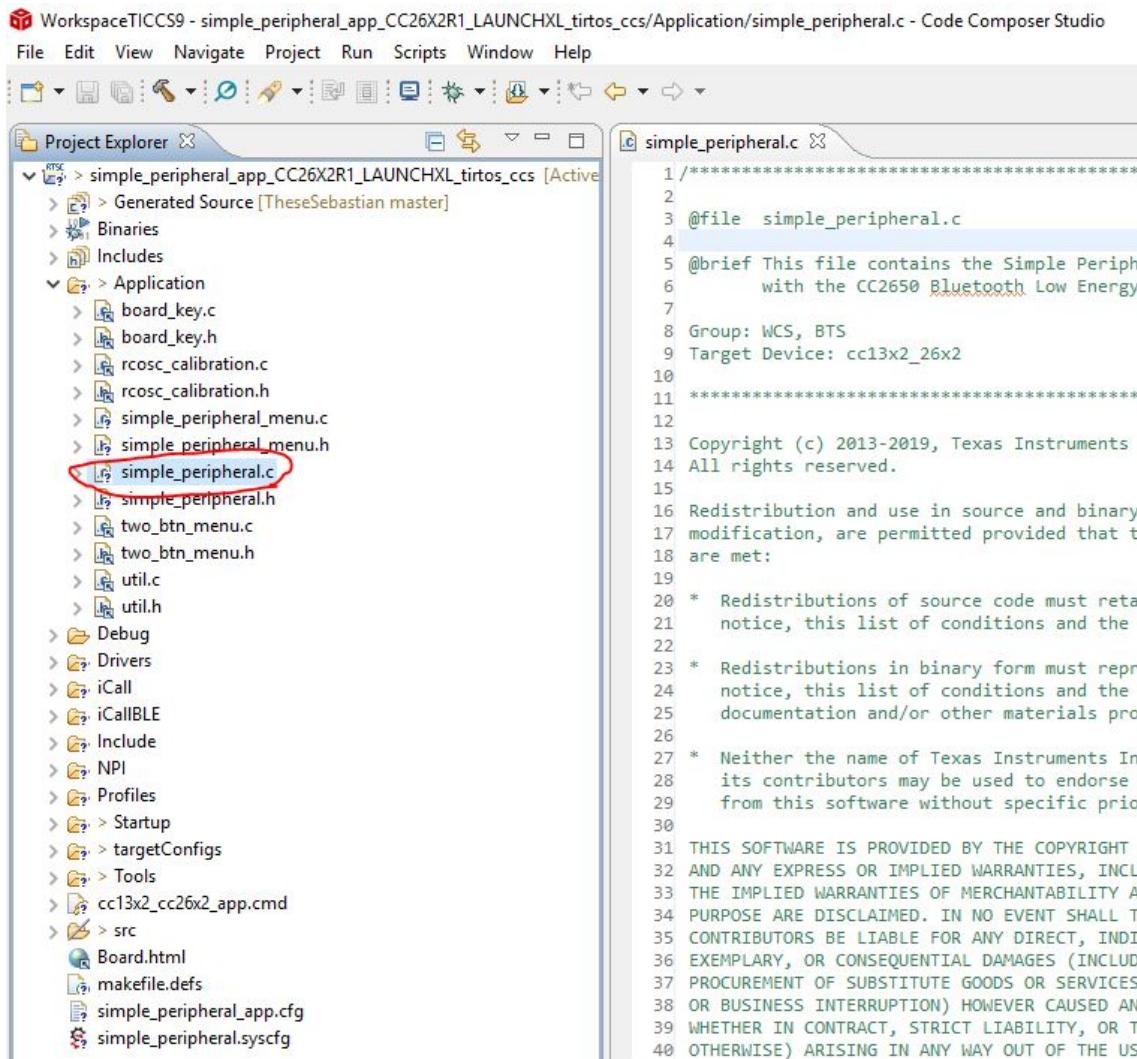


FIGURE 26 – Crédit de la création du profile

En sous de l'enum *Auto connect available groups* ajouter :

```
uint8_t BufGRX[20]={0};

// Struct for messages about characteristic data
typedef struct
{
    uint16_t svcUUID; // UUID of the service
    uint16_t dataLen;
    uint8_t paramID; // Index of the characteristic
    uint8_t data[]; // Flexible array member,
```

```

//extended to malloc - sizeof(.)
} pzCharacteristicData_t;

// Declaration of service callback handlers
static void user_Accelerometre_ValueChangeCB(
    uint16_t connHandle,
    uint8_t paramID,
    uint16_t len,
    uint8_t *pValue); // Callback from the service.
static void user_Accelerometre_ValueChangeHandler(
    pzCharacteristicData_t *pCharData); // Local handler
//called from the Task context of this task.

// Service callback function implementation
// Accelerometre callback handler.
//The type AccelerometreCBs_t is defined in Accelerometre.h
static AccelerometreCBs_t user_AccelerometreCBs =
{.pfnChangeCb = user_Accelerometre_ValueChangeCB,
//Characteristic value change callback handler
//.pfnCfgChangeCb = NULL, // No CCCD change handler implemented
};


```

Juste en sous de `setBondManagerParameters()`; comme montre la figure 27 on va ajouter la déclaration du service :

```

WorkspaceTICCS9 - simple_peripheral_app_CC26X2R1_LAUNCHXL_tirtos_ccs/Application/simple_peripheral.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer simple_peripheral.c ti_ble_config.c
690 // Configure GAP
691 {
692     uint16_t paramUpdateDecision = DEFAULT_PARAM_UPDATE_REQ_DECISION;
693
694     // Pass all parameter update requests to the app for it to decide
695     GAP_SetParamValue(GAP_PARAM_LINK_UPDATE_DECISION, paramUpdateDecision);
696 }
697
698 // Setup the GAP Bond Manager. For more information see the GAP Bond Manager
699 // section in the User's Guide
700 setBondManagerParameters();
701
702 /*TcJ, Here, Aquia*/
703
704 // Initialize GATT attributes
705 GGS_AddService(GATT_ALL_SERVICES); // GAP GATT Service
706 GATTServerApp_AddService(GATT_ALL_SERVICES); // GATT Service
707 DevInfo_AddService(); // Device Information Service
708 SimpleProfile_AddService(GATT_ALL_SERVICES); // Simple GATT Profile
709
710
711 // Setup the SimpleProfile Characteristic Values
712 // For more information, see the GATT and GATTServerApp sections in the User's Guide:
713 // http://software-dl.ti.com/lprf/ble5stack-latest/
714 {
715     uint8_t charValue1 = 1;
716     uint8_t charValue2 = 2;
717     uint8_t charValue3 = 3;
718     uint8_t charValue4 = 4;
719     uint8_t charValue5[SIMPLEPROFILE_CHARS_LEN] = { 1, 2, 3, 4, 5 };
720
721     SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR1, sizeof(uint8_t),
722                               &charValue1);
723     SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR2, sizeof(uint8_t),
724                               &charValue2);
725     SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR3, sizeof(uint8_t),
726                               &charValue3);
727     SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR4, sizeof(uint8_t),
728                               &charValue4);
729     SimpleProfile_SetParameter(SIMPLEPROFILE_CHAR5, SIMPLEPROFILE_CHARS_LEN,

```

FIGURE 27 – Création du profile

```

Accelerometre_AddService();
Accelerometre_RegisterAppCBs(&user_AccelerometreCBs);

```

```
// Initialization of characteristics in
// Accelerometre that are readable.

uint8_t Accelerometre_AccelerometreMesures_initVal[
ACCELEROMETRE_ACCELEROMETREMESURES_LEN] = {0};
Accelerometre_SetParameter(
ACCELEROMETRE_ACCELEROMETREMESURES,
ACCELEROMETRE_ACCELEROMETREMESURES_LEN,
Accelerometre_AccelerometreMesures_initVal);
```

Juste en bas de *case SP_CONN_EVT* : ajouter

```
case PZ_MSG_ACCELEROMETRE:
    SendAccelerometreMesure();
    break;
```

A la fin du fichier *simple_peripheral.c* on va ajouter les fonctions suivantes :

```
void user_Accelerometre_ValueChangeHandler(
pzCharacteristicData_t *pData)
{
    switch (pData->paramID)
    {
        case ACCELEROMETRE_ACCELEROMETREMESURES:
            //Log_info0("Value Change msg for Accelerometre
            //:: AccelerometreMesures received");
            // Do something useful with pData->data here
            // -----
            break;
    }
}

static void user_Accelerometre_ValueChangeCB(uint16_t connHandle,
                                             uint8_t paramID, uint16_t len,
                                             uint8_t *pValue)
{
    pzCharacteristicData_t *pValChange =
    ICall_malloc(sizeof(pzCharacteristicData_t) + len);

    if(pValChange != NULL)
    {
        pValChange->svcUUID = ACCELEROMETRE_SERV_UUID;
        pValChange->paramID = paramID;
        memcpy(pValChange->data, pValue, len);
        pValChange->dataLen = len;

        SimplePeripheral_enqueueMsg(SP_STATE_CHANGE_EVT,
                                    pValChange);
    }
}
```

```

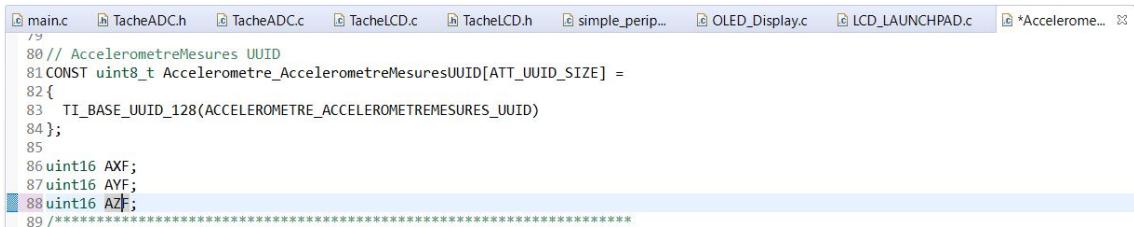
void Carte_enqueueMsg(uint8_t event){
    spEvt_t *pMsg = ICall_malloc(sizeof(spEvt_t));

    // Create dynamic pointer to message.
    if(pMsg)
    {
        pMsg->event = event;
        pMsg->pData = NULL;

        // Enqueue the message.
        Util_enqueueMsg(appMsgQueueHandle,
                        syncEvent, (uint8_t *)pMsg);
    }
}

```

La fonction *SendAccelerometreMesure* il faut la faire dans le fichier source de notre service (*Accelerometre.c*). On va déclarer comme variables globales toutes celles qui ne sont pas définies Comme montre l'image 28, et dans la prochaine section on fera la fonction.



The screenshot shows a code editor with multiple tabs at the top: main.c, TacheADC.h, TacheADC.c, TacheLCD.c, TacheLCD.h, simple_perip..., OLED_Display.c, LCD_LAUNCHPAD.c, and *Accelerome... . The code in the editor is:

```

79
80 // AccelerometreMesures UUID
81 CONST uint8_t Accelerometre_AccelerometreMesuresUUID[ATT_UUID_SIZE] =
82 {
83     TI_BASE_UUID_128(ACCELEROMETRE_ACCELEROMETREMESURES_UUID)
84 };
85
86 uint16 AXF;
87 uint16 AVF;
88 uint16 AZF;
89 ****

```

FIGURE 28 – Crédit de la figure

3.8 Envoi des données par BLE

Pour l'envoie de données, tout d'abord dans la tâche ADC on va appeler une fonction qui sera créée dans le fichier source du service de l'accéléromètre pour enregistrer les données qu'on vient d'obtenir dans les variables globales déclarés précédemment. Dans la figure 29 on a la fonction *SendAccelerometerMesure* et la fonction *saveDataToSend*. Cette dernière accompagnée d'une autre fonction vont être appelées dans la *TacheADC*.

```

main.c TacheADC.h *TacheADC.c TacheLCD.c simple_perip... LCD_LAUNCHPAD.c Accelerometre.c
387
388     return status;
389 }
390
391 void SendAccelerometreMesure(void){
392     uint8_t indexArr=0;
393     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0xFE;
394     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
395     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x70;
396     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
397     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x0E;
398     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
399     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
400     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AXF >>8);
401     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AXF);
402     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AYF >>8);
403     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AYF);
404     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AZF >>8);
405     Accelerometre_AccelerometreMesuresVal[indexArr++] = (uint8_t)(AZF);
406     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
407     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
408     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
409     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
410     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
411     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
412     Accelerometre_AccelerometreMesuresVal[indexArr++] = 0x00;
413     Accelerometre_SetParameter(ACCELEROMETRE_ACCELEROMETREMESURES,
414                                 ACCELEROMETRE_ACCELEROMETREMESURES_LEN,
415                                 Accelerometre_AccelerometreMesuresVal);
416 }
417 void SaveDataToSend(float AxADC, float AyADC, float AzADC){
418     AXF = (uint16) AxADC;
419     AYF = (uint16) AyADC;
420     AZF = (uint16) AzADC;
421 }
...

```

FIGURE 29 – Création du profile

Ne pas oublier d'ajouter les fonctions au fichier .h correspondant

La fonction `Carte_enqueueMsg` sera appelée dans la `TacheADC` juste après de la fonction `saveDatatoSend`, la variable `event` sera : `ACCELEROMETRE_SERV_UUID`. Si on compile on verra que le compilateur nous dit *integer conversion resulted in truncation* Donc on va définir dans le fichier header du service BLE le suivant :

```
#define PZ_MSG_ACCELEROMETRE 10
```

Ce notre evenement associé à l'envoie des données de l'accelerometre. Donc il faudra aller sur la fonction qui processe les messages de l'application dans le fichier `simple_peripheral.c` et changer : `ACCELEROMETRE_SERV_UUID` par `PZ_MSG_ACCELEROMETRE` de cette façon les fonctions appelée dans la `TacheADC` seront celles qui montre la figure 30

```

84
85
86 static void TacheADC_taskFxn(UArg a0, UArg a1)
87 {
88     // Initialize application
89     TacheADC_init();
90     // Application main loop
91     for (;;)
92     {
93         Semaphore_pend(semADCHandle, BIOS_WAIT_FOREVER);
94
95         uint32_t DatasampledX = Sampling(Board_ADC5);
96         uint32_t DatasampledY = Sampling(Board_ADC3);
97         uint32_t DatasampledZ = Sampling(Board_ADC4);
98
99         float AccX = uVToG_float(DatasampledX);
100        float AccY = uVToG_float(DatasampledY);
101        float AccZ = uVToG_float(DatasampledZ);
102
103        saveDatatoSend(AccX, AccY, AccZ);
104        Carte_enqueueMsg(PZ_MSG_ACCELEROMETRE, NULL);
105        afficherDonnées(AccX, AccY, AccZ);
106    }
107 }
108

```

FIGURE 30 – Création du profile

Ajouter les lignes suivants au fichier Accelerometre.h :

```
#include "bcomdef.h"
#include "_hal_types.h"
```

Commenter les lignes comme montre la Fig. 31 et ajouter *return 0 ;* à la fin.

```

main.c      TacheADC.c    TacheLCD.c    *simple_peri...  Accelerometre.c  simple_perip...
1827 * @fn      SimplePeripheral_enqueueMsg
1828 *
1829 * @brief   Creates a message and puts the message in RTOS queue.
1830 *
1831 * @param   event - message event.
1832 * @param   state - message state.
1833 */
1834 static status_t SimplePeripheral_enqueueMsg(uint8_t event, void *pData)
1835 {
1836 //  uint8_t success;
1837 //  spEvt_t *pMsg = ICall_malloc(sizeof(spEvt_t));
1838 //
1839 //  // Create dynamic pointer to message.
1840 //  if(pMsg)
1841 //  {
1842 //      pMsg->event = event;
1843 //      pMsg->pData = pData;
1844 //
1845 //      // Enqueue the message.
1846 //      success = Util_enqueueMsg(appMsgQueueHandle, syncEvent, (uint8_t *)pMsg);
1847 //      return (success) ? SUCCESS : FAILURE;
1848 //  }
1849 //
1850 //  return(bleMemAllocError);
1851 return 0;
1852 }

```

FIGURE 31 – Commenter lignes pour éviter problèmes de compatibilité avec l'exemple de base

Si le capteur ne marche pas, il faut regarder les priorités des tâches. La tâche LCD doit avoir la priorité mineur (par exemple 1), puis la tâche ADC une priorité de 1 et la tâche *simple_peripheral* pareil que la dernier donc 1

4 Joystick

Ajouter au projet la gestion du joystick et l'envoie des données par BLE. Pour cela il faudra :

1. Ajouter 2 channels ADC pour échantillonner les 2 sorties du joystick (Vertical : J3.26 et Horizontal J1.2)
2. Echantillonner les sorties du joystick à 50 Hz (Tâche ADC)
3. Envoyer les données à la tâche LCD et afficher la valeur
4. Ajouter un service BLE pour envoyer les données du joystick

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);  
}
```