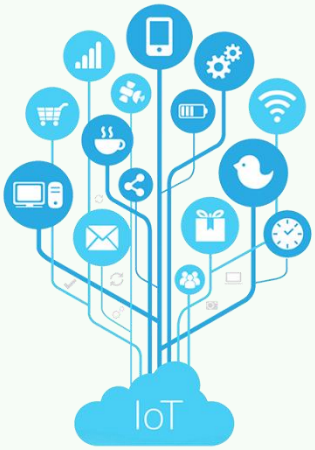
UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

TP n°3 : Solutions

MASTER RÉSEAUX ET TÉLÉCOM



VERSION : 10 juin 2021

Florent NOLOT
UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Table des matières

I. Solutions

A. Exercice 1 – Capteurs

1. Récupérer et afficher les données de température et de pression

Modification du fichier main

```
/* TI-RTOS Header files */
#include <ti/drivers/PIN.h>
#include <ti/drivers/I2C.h>
#include "SensorBmp280.h"
#include "SensorI2C.h"
```

```
Void taskFxn(UArg arg0, UArg arg1){

    int i=0;
    uint8_t data;
    int32_t temp;
    uint32_t press;

    if( (SensorI2C_open()) == false ){
        System_abort("Error opening I2C \n");System_flush();
    }
    if( (SensorBmp280_init()) == false ){
        System_abort("Error init BMP280\n");System_flush();
    }

    while(1){
        i++;

        SensorBmp280_enable(true);

        if( (SensorBmp280_read(&data)) == false){
            System_abort("Error read BMP280\n");System_flush();
        }

        SensorBmp280_convert(&data, &temp, &press);

        System_printf("\nRead %d : temp = %d ; press = %d",i,temp, press);System_flush();

        Task_sleep(100000 / Clock_tickPeriod);

    }
```

2. Récupérer et afficher les données d'humidité

Modification du fichier main

```
#include "SensorI2C.h"
#include "SensorHdc1000.h"
```

```
Void heartBeatFxn(UArg arg0, UArg arg1)
{
    int i=0;
    uint16_t rawTemp, rawHum;
    float temp, hum;

    if( SensorI2C_open() == false ){System_abort("Error open I2C\n");System_flush();}

    SensorHdc1000_test();

    if( SensorHdc1000_init() == false ){System_abort("Error init Sensor Hdc\n");System_flush();}


    while(1){
        SensorHdc1000_start();
        Task_sleep(200000 / Clock_tickPeriod);
        if( SensorHdc1000_read(&rawTemp, &rawHum) == false ){System_abort("Error read Hdc\n"); System_flush();}
        SensorHdc1000_convert(rawTemp, rawHum, &temp, &hum);

        System_printf("Read %d : Temp = %f ; Hum = %f\n", i, temp, hum);System_flush();
        Task_sleep(3000000 / Clock_tickPeriod);
    }
}
```

3. Récupérer et afficher les données de luminosité

Modification du fichier main

```
/* TI-RTOS Header files */
#include <ti/drivers/PIN.h>
#include <ti/drivers/I2C.h>
#include "SensorI2C.h"
#include "SensorOpt3001.h"
```

```
Void heartBeatFxn(UArg arg0, UArg arg1){
    float lux=0;
    uint16_t rawData;

    if( SensorI2C_open() == false ){System_abort("Error opening I2C"); System_flush();}
    if( SensorOpt3001_init() == false ){System_abort("Error init sensor opt3001");System_flush();}


    while(1){
        SensorOpt3001_enable(true);
        if( SensorOpt3001_read(&rawData) == false ){System_abort("Error read data sensor opt3001");System_flush();}
        System_printf("rawData : %d\n",rawData);System_flush();
        lux = SensorOpt3001_convert(rawData);

        System_printf("lux : %f\n",lux);System_flush();

        Task_sleep(3000000 / Clock_tickPeriod);
    }


}
```

4. Récupérer et afficher les données de force, de vitesse angulaire et d'électromagnétisme suivant les 3 axes (x,y,z)

Modification du fichier main

```
#include "SensorI2C.h"
#include "SensorMpu9250.h"
```

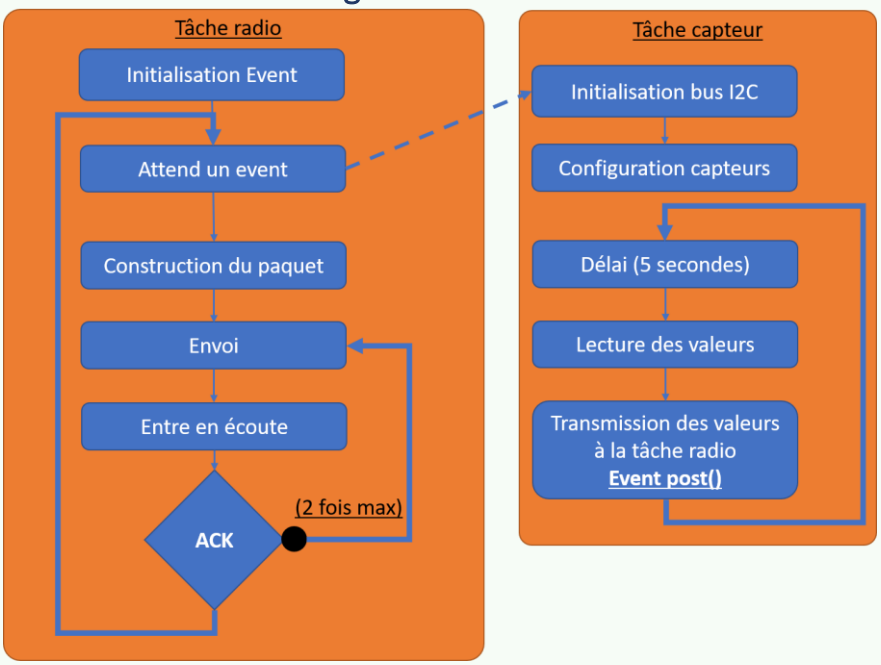
```
Void heartBeatFxn(UArg arg0, UArg arg1){
    uint16_t rawDataAcc[3];
    uint16_t rawDataGyro[3];
    int16_t rawDataMag[3];
    uint16_t x_acc, y_acc, z_acc;
    float x_gyro, y_gyro, z_gyro;
    int16_t x_mag, y_mag, z_mag;


    /*** Initialization MPU9250 ***/
    if(SensorI2C_open()          == false){System_abort("Error open I2C\n");System_flush();}
    if(SensorMpu9250_init()      == false){System_abort("Error init sensor mpu 9250\n");System_flush();}
    SensorMpu9250_powerOn();
    if(SensorMpu9250_powerIsOn() == false){System_abort("Error sensor mpu9250 is off\n");System_flush();}
    if(SensorMpu9250_test()      == false){System_abort("Error sensor test\n");System_flush();}
    if(SensorMpu9250_magTest()   == false){System_abort("Error mag test\n");System_flush();}
    if(SensorMpu9250_magStatus() != MAG_STATUS_OK){System_abort("Error mag status not ok\n");System_flush();}
    if(SensorMpu9250_accSetRange(ACC_RANGE_2G) == false){System_abort("Error set range acc\n");System_flush();}

    SensorMpu9250_enable(255);
```

```
    /*** Read data ***/
    while(1){

        /* Read Acc */
        if(SensorMpu9250_accRead(rawDataAcc) == false){System_abort("Error Read\n");System_flush();}
        x_acc = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[0])) ));
        y_acc = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[1])) ));
        z_acc = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[2])) ));

        /* Read Gyro */
        if(SensorMpu9250_gyroRead(rawDataGyro) == false ){System_abort("Error read gyro\n");System_flush();}
        x_gyro = SensorMpu9250_gyroConvert(rawDataGyro[0]);
        y_gyro = SensorMpu9250_gyroConvert(rawDataGyro[1]);
        z_gyro = SensorMpu9250_gyroConvert(rawDataGyro[2]);

        /* Read Mag */
        if(SensorMpu9250_magRead(rawDataMag) == MAG_STATUS_OK){
            x_mag = rawDataMag[0];
            y_mag = rawDataMag[1];
            z_mag = rawDataMag[2];
        }

        System_printf("\nAcc   : X  = %d  ; Y  = %d  ; Z  = %d\n", x_acc, y_acc, z_acc);
        System_printf("Gyro  : vX = %f  ; vY = %f  ; vZ = %f\n",x_gyro, y_gyro, z_gyro);
        System_printf("Mag   : X  = %d  ; Y  = %d  ; Z  = %d\n", x_mag, y_mag, z_mag);

        System_flush();

        Task_sleep(2000000 / Clock_tickPeriod);
    }
}
```

B. Exercice 2 – Application IOT

A l'aide des TPs précédents transférez les données de récupérées à un autre Sensor Tag (le concentrateur) qui les envoie sur le port série en format JSON.

1. Programme d'émission



Le programme d'un nœud initialise deux tâches, une tâche radio (NodeRadioTask.c) et une tâche capteur (NodeTask.c) qui récupère les informations des capteurs et une tâche radio qui gère la transmission des données au concentrateur.

1. La tâche radio est initialisée en premier, la tâche configure les propriétés réseaux du nœud et crée un évènement positionné à la valeur 0.
2. La tâche radio se positionne en mode bloqué sur l'évènement.
3. La tâche radio étant en mode bloqué, la tâche capteur s'exécute pour la première fois, elle ouvre la communication I2C et configure les capteurs en transmettant des valeurs précises dans les registres de configuration de chaque capteurs.
4. La tâche capteur se positionne en attente pendant 5 secondes (choisie arbitrairement), les deux tâches sont désormais bloquées et la tâche Idle s'exécute.
5. A l'issue du délai le tâche capteur récupère une données à la fois pour chaque capteur
6. La tâche capteur transmet les valeurs récupérées à la tâche radio et incrémente d'une unité l'évènement.
7. L'évènement étant incrémenté d'une unité, la tâche radio s'exécute et construit le paquet et décrémente d'une unité l'évènement.
8. La tâche radio envoie le paquet et entre mode en écoute.
9. Si la tâche radio reçoit un acquittement de la part du concentrateur elle se re positionne en attente, dans le cas contraire elle envoie à nouveau le paquet, toutefois si elle a déjà envoyé deux fois le même paquet, la tâche radio se positionne en attente.

Modification du fichier « NodeTask.c »

```
#include <ti/drivers/I2C.h>
#include "sensorsHeaders/SensorI2C.h"
#include "sensorsHeaders/SensorBmp280.h"
#include "sensorsHeaders/SensorOpt3001.h"
#include "sensorsHeaders/SensorMpu9250.h"
#include "sensorsHeaders/SensorHdc1000.h"
#include <math.h>
#include <string.h>
```

```
/***** GLOBAL VARIABLES *****/

/*** Data & RawData sensors ***/
static sensors_data data;
static uint8_t rawDataBmp;
static uint16_t rawTempHDC;
static float tempHDC;
static uint16_t rawHumHDC;
static uint16_t rawDataOpt;
static uint16_t rawDataAcc[3];
static uint16_t rawDataAnVel[3];
static int16_t rawDataMagFd[3];
```

```
static void nodeTaskFunction(UArg arg0, UArg arg1)
{
    int32_t temp_int=0;

    /* Initializes UART Display */
    Display_Params params;
    Display_Params_init(&params);
    params.lineClearMode = DISPLAY_CLEAR_BOTH;
    //hDisplaySerial = Display_open(Display_Type_UART, &params);
    /* Check if the selected Display type was found and successfully opened */
    if (hDisplaySerial){Display_printf(hDisplaySerial, 0, 0, "Waiting for Sensors ADC reading...");}

    /* Initializes PINs */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (!ledPinHandle){System_abort("Error initializing board 3.3V domain pins\n");}

    /* Initializes I2C bus & Sensors */
    if(SensorI2C_open()          == false ){System_printf("Error opening I2C \n");System_flush();}
    if(SensorBmp280_init()       == false ){System_printf("Error init BMP280\n");System_flush();}
    SensorHdc1000_test();
    if(SensorHdc1000_init()      == false ){System_printf("Error init Sensor Hdc\n");System_flush();}
    if(SensorOpt3001_init()      == false ){System_printf("Error init sensor opt3001");System_flush();}
    if(SensorMpu9250_init()      == false){System_printf("Error init sensor mpu 9250\n");System_flush();}
    SensorMpu9250_powerOn();
    if(SensorMpu9250_powerIsOn() == false){System_printf("Error sensor mpu9250 is off\n");System_flush();}
    if(SensorMpu9250_test()      == false){System_printf("Error sensor test\n");System_flush();}
    if(SensorMpu9250_magTest()   == false){System_printf("Error mag test\n");System_flush();}
    if(SensorMpu9250_magStatus() != MAG_STATUS_OK){System_printf("Error mag status not ok\n");System_flush();}
    if(SensorMpu9250_accSetRange(ACC_RANGE_2G) == false){System_printf("Error set range acc\n");System_flush();}
    SensorHdc1000_test();
    SensorMpu9250_enable(255);
    SensorBmp280_enable(true);
    SensorOpt3001_enable(true);
```

```
while (1){

    /* Wait 3s */
    Task_sleep(10000000 / Clock_tickPeriod);

    /* Sensor BMP 280 : Read Temperature & pressure */
    if( (SensorBmp280_read(&rawDataBmp)) == false){System_printf("Error read BMP280\n");System_flush();}
    SensorBmp280_convert(&rawDataBmp, &temp_int, &data.press);
    data.temp = (((float)temp_int) * (powf(10,-2))) - (6.0);

    /* Sensor HDC 1000 : Read Humidity and Temperature */
    SensorHdc1000_start();
    if( SensorHdc1000_read(&rawTempHDC, &rawHumHDC) == false ){System_printf("Error read Hdc\n"); System_flush();}
    SensorHdc1000_convert(rawTempHDC, rawHumHDC, &tempHDC, &data.hum);

    /* Sensor OPT 3001 : Read Luminosity */
    if( SensorOpt3001_read(&rawDataOpt) == false ){System_printf("Error read data sensor opt3001");System_flush();}
    data.lum = SensorOpt3001_convert(rawDataOpt);

    /* Sensor MPU9250 : Read Acceleration (x ; y ; z) */
    if(SensorMpu9250_accRead(rawDataAcc) == false){System_abort("Error Read\n");System_flush();}
    data.acc[0] = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[0])) ));
    data.acc[1] = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[1])) ));
    data.acc[2] = (uint16_t) (fabsf( (SensorMpu9250_accConvert(rawDataAcc[2])) ));

    /* Sensor MPU9250 : Read Angular Velocity (x ; y ; z) */
    if(SensorMpu9250_gyroRead(rawDataAnVel) == false ){System_abort("Error read gyro\n");System_flush();}
    data.anVel[0] = SensorMpu9250_gyroConvert(rawDataAnVel[0]);if(data.anVel[0] <= 5)data.anVel[0]=0;
    data.anVel[1] = SensorMpu9250_gyroConvert(rawDataAnVel[1]);if(data.anVel[1] <= 5)data.anVel[1]=0;
    data.anVel[2] = SensorMpu9250_gyroConvert(rawDataAnVel[2]);if(data.anVel[2] <= 5)data.anVel[2]=0;


    /* Sensor MPU9250 : Read Magnetic Field (x ; y ; z) */
    if(SensorMpu9250_magRead(rawDataMagFd) == MAG_STATUS_OK){
        data.magFd[0] = rawDataMagFd[0];
        data.magFd[1] = rawDataMagFd[1];
        data.magFd[2] = rawDataMagFd[2];
    }
```

```
        /* Toggle activity LED */
        PIN_setOutputValue(ledPinHandle, NODE_ACTIVITY_LED,!PIN_getOutputValue(NODE_ACTIVITY_LED));

        NodeRadioTask_sendAdcData(&data);
    }
}
```

Modification du fichier « RadioProtocol.h »

```
struct DualModeSensorPacket {
    struct PacketHeader header;
    uint32_t temperature;
    uint32_t pressure;
    uint32_t humidity;
    uint32_t luminosity;
    uint32_t acceleration[3];
    uint32_t angularVelocity[3];
    uint32_t magneticField[3];
    uint32_t time100MiliSec;
    uint16_t batt;
};
```

Modification du fichier « NodeRadioTask.c »

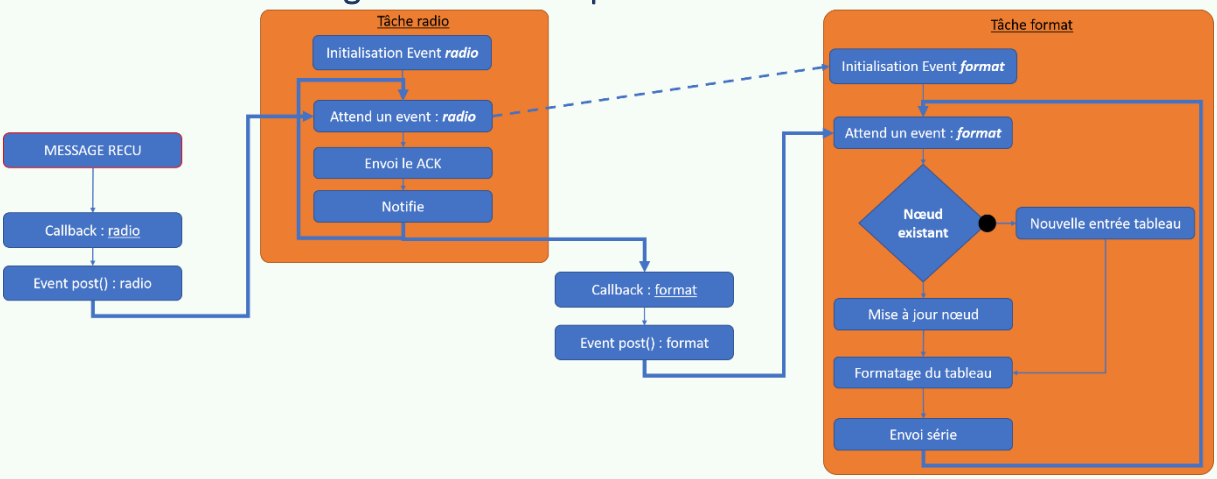
```
/***** Global variables : DATA *****/
static uint32_t temperature;
static uint32_t pressure;
static uint32_t humidity;
static uint32_t luminosity;
static uint32_t acceleration[3];
static uint32_t angularVelocity[3];
static uint32_t magneticField[3];
```

```
/* Enter main task loop */
while (1)
{
    /* Wait for an event */
    uint32_t events = Event_pend(radioOperationEventHandle, 0, RADIO_EVENT_ALL, BIOS_WAIT_FOREVER);

    /* If we should send ADC data */
    if (events & RADIO_EVENT_SEND_ADC_DATA)
    {
        uint32_t currentTicks;

        currentTicks = Clock_getTicks();
        //check for wrap around
        if (currentTicks > prevTicks)
        {
            //calculate time since last reading in 0.1s units
            dmSensorPacket.time100MiliSec += ((currentTicks - prevTicks) * Clock_tickPeriod) / 100000;
        }
        else
        {
            //calculate time since last reading in 0.1s units
            dmSensorPacket.time100MiliSec += ((prevTicks - currentTicks) * Clock_tickPeriod) / 100000;
        }
        prevTicks = currentTicks;

        dmSensorPacket.batt = AONBatMonBatteryVoltageGet();
        dmSensorPacket.temperature = temperature;
        dmSensorPacket.pressure = pressure;
        dmSensorPacket.humidity = humidity;
        dmSensorPacket.luminosity = luminosity;
        dmSensorPacket.acceleration[0] = acceleration[0];
        dmSensorPacket.acceleration[1] = acceleration[1];
        dmSensorPacket.acceleration[2] = acceleration[2];
        dmSensorPacket.angularVelocity[0] = angularVelocity[0];
        dmSensorPacket.angularVelocity[1] = angularVelocity[1];
        dmSensorPacket.angularVelocity[2] = angularVelocity[2];
        dmSensorPacket.magneticField[0] = magneticField[0];
        dmSensorPacket.magneticField[1] = magneticField[1];
        dmSensorPacket.magneticField[2] = magneticField[2];

        sendDmPacket(dmSensorPacket, NODERADIO_MAX_RETRIES, NORERADIO_ACK_TIMEOUT_TIME_MS);
```

```
enum NodeRadioOperationStatus NodeRadioTask_sendAdcData(sensors_data *data)
{
    enum NodeRadioOperationStatus status;

    /* Get radio access sempahore */
    Semaphore_pend(radioAccessSemHandle, BIOS_WAIT_FOREVER);

    /* Save data to send */
    pressure            =   (uint32_t) data->press;
    acceleration[0]     =   (uint32_t) data->acc[0];
    acceleration[1]     =   (uint32_t) data->acc[1];
    acceleration[2]     =   (uint32_t) data->acc[2];
    magneticField[0]    =   (uint32_t) data->magFd[0];
    magneticField[1]    =   (uint32_t) data->magFd[1];
    magneticField[2]    =   (uint32_t) data->magFd[2];
    memcpy(&temperature, &data->temp, sizeof(data->temp));
    memcpy(&humidity, &data->hum, sizeof(data->hum));
    memcpy(&luminosity, &data->lum, sizeof(data->lum));
    memcpy(&angularVelocity[0], &data->anVel[0], sizeof(uint32_t));
    memcpy(&angularVelocity[1], &data->anVel[1], sizeof(uint32_t));
    memcpy(&angularVelocity[2], &data->anVel[2], sizeof(uint32_t));

    /* Raise RADIO_EVENT_SEND_ADC_DATA event */
    Event_post(radioOperationEventHandle, RADIO_EVENT_SEND_ADC_DATA);

    /* Wait for result */
    Semaphore_pend(radioResultSemHandle, BIOS_WAIT_FOREVER);
```

```
static void sendDmPacket(struct DualModeSensorPacket sensorPacket, uint8_t maxNumberOfRetries, uint32_t ackTimeoutMs)
{

    /* Set destination address in EasyLink API */
    currentRadioOperation.easyLinkTxPacket.dstAddr[0] = RADIO_CONCENTRATOR_ADDRESS;

    /* Copy ADC packet to payload
     * Note that the EasyLink API will implcitily both add the length byte and the destination address byte. */
    currentRadioOperation.easyLinkTxPacket.payload[0] = dmSensorPacket.header.sourceAddress;
    currentRadioOperation.easyLinkTxPacket.payload[1] = dmSensorPacket.header.packetType;

    currentRadioOperation.easyLinkTxPacket.payload[2] = (dmSensorPacket.temperature & 0xff000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[3] = (dmSensorPacket.temperature & 0x00ff0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[4] = (dmSensorPacket.temperature & 0x0000ff00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[5] = (dmSensorPacket.temperature & 0x000000ff);

    currentRadioOperation.easyLinkTxPacket.payload[6] = (dmSensorPacket.pressure & 0xff000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[7] = (dmSensorPacket.pressure & 0x00ff0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[8] = (dmSensorPacket.pressure & 0x0000ff00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[9] = (dmSensorPacket.pressure & 0x000000ff);

    currentRadioOperation.easyLinkTxPacket.payload[10] = (dmSensorPacket.humidity & 0xff000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[11] = (dmSensorPacket.humidity & 0x00ff0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[12] = (dmSensorPacket.humidity & 0x0000ff00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[13] = (dmSensorPacket.humidity & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[14] = (dmSensorPacket.luminosity & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[15] = (dmSensorPacket.luminosity & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[16] = (dmSensorPacket.luminosity & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[17] = (dmSensorPacket.luminosity & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[18] = (dmSensorPacket.acceleration[0] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[19] = (dmSensorPacket.acceleration[0] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[20] = (dmSensorPacket.acceleration[0] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[21] = (dmSensorPacket.acceleration[0] & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[22] = (dmSensorPacket.acceleration[1] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[23] = (dmSensorPacket.acceleration[1] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[24] = (dmSensorPacket.acceleration[1] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[25] = (dmSensorPacket.acceleration[1] & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[26] = (dmSensorPacket.acceleration[2] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[27] = (dmSensorPacket.acceleration[2] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[28] = (dmSensorPacket.acceleration[2] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[29] = (dmSensorPacket.acceleration[2] & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[30] = (dmSensorPacket.angularVelocity[0] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[31] = (dmSensorPacket.angularVelocity[0] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[32] = (dmSensorPacket.angularVelocity[0] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[33] = (dmSensorPacket.angularVelocity[0] & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[34] = (dmSensorPacket.angularVelocity[1] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[35] = (dmSensorPacket.angularVelocity[1] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[36] = (dmSensorPacket.angularVelocity[1] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[37] = (dmSensorPacket.angularVelocity[1] & 0x000000ff);

  currentRadioOperation.easyLinkTxPacket.payload[38] = (dmSensorPacket.angularVelocity[2] & 0xff000000) >> 24;
  currentRadioOperation.easyLinkTxPacket.payload[39] = (dmSensorPacket.angularVelocity[2] & 0x00ff0000) >> 16;
  currentRadioOperation.easyLinkTxPacket.payload[40] = (dmSensorPacket.angularVelocity[2] & 0x0000ff00) >> 8;
  currentRadioOperation.easyLinkTxPacket.payload[41] = (dmSensorPacket.angularVelocity[2] & 0x000000ff);

    currentRadioOperation.easyLinkTxPacket.payload[46] = (dmSensorPacket.magneticField[1] & 0xff000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[47] = (dmSensorPacket.magneticField[1] & 0x00ff0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[48] = (dmSensorPacket.magneticField[1] & 0x0000ff00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[49] = (dmSensorPacket.magneticField[1] & 0x000000ff);

    currentRadioOperation.easyLinkTxPacket.payload[50] = (dmSensorPacket.magneticField[2] & 0xff000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[51] = (dmSensorPacket.magneticField[2] & 0x00ff0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[52] = (dmSensorPacket.magneticField[2] & 0x0000ff00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[53] = (dmSensorPacket.magneticField[2] & 0x000000ff);

    currentRadioOperation.easyLinkTxPacket.payload[54] = (dmSensorPacket.batt & 0xFF00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[55] = (dmSensorPacket.batt & 0xFF);
    currentRadioOperation.easyLinkTxPacket.payload[56] = (dmSensorPacket.time100MiliSec & 0xFF000000) >> 24;
    currentRadioOperation.easyLinkTxPacket.payload[57] = (dmSensorPacket.time100MiliSec & 0x00FF0000) >> 16;
    currentRadioOperation.easyLinkTxPacket.payload[58] = (dmSensorPacket.time100MiliSec & 0xFF00) >> 8;
    currentRadioOperation.easyLinkTxPacket.payload[59] = (dmSensorPacket.time100MiliSec & 0xFF);

    currentRadioOperation.easyLinkTxPacket.len = sizeof(struct DualModeSensorPacket);
```

2. Programme de réception



Le programme du concentrateur est composé de deux tâches, une tâche permettant de gérer la réception de paquet provenant des nœuds et l'envoi d'acquittement (ConcentratorRadioTask.c) et une autre tâche (ConcentratorTask.c) permettant la gestion du formatage des données en format JSON et l'envoie des ces données formatées sur la liaison série.

1. La tâche radio s'exécute en premier et initialise un évènement *Radio* à 0.
2. La tâche radio se met en attente sur l'évènement *Radio.*
3. La tâche format s'exécute ensuite et initialise un évènement *Format* et se met en attente sur l'évènement *Format*, les deux tâches sont alors en mode bloquées.
4. Lors de la réception d'un message de la part d'un nœud l'évènement *Radio* est incrémentée d'une unité et envoie un acquittement, ainsi la tâche radio transmet à la tâche format et incrémente d'une unité l'événement *Format* ce qui a pour conséquence de débloquée la tâche format.
5. La tâche format compare l'adresse du nœud du paquet reçu avec celles qui possède déjà, si l'adresse du paquet reçu n'existe pas, une nouvelle entrée est créée dans le cas contraire, les informations du nœuds sont mis à jour.

La tâche format formate toutes les informations de chaque nœuds en en format JSON et l'envoie sur la liaison série.

Modification du fichier « RadioProtocol.h »

```
struct DualModeSensorPacket {
    struct PacketHeader header;
    uint32_t temperature;
    uint32_t pressure;
    uint32_t humidity;
    uint32_t luminosity;
    uint32_t acceleration[3];
    uint32_t angularVelocity[3];
    uint32_t magneticField[3];
    uint32_t time100MiliSec;
    uint16_t batt;
};
```

Modification du fichier « ConcentrateurRadioTask.c »

- Fonction rxDoneCallback

```
else if (tmpRxPacket->header.packetType == RADIO_PACKET_TYPE_DM_SENSOR_PACKET)
{
    /* Save packet */
    latestRxPacket.header.sourceAddress = rxPacket->payload[0];
    latestRxPacket.header.packetType = rxPacket->payload[1];

    latestRxPacket.dmSensorPacket.temperature = (rxPacket->payload[2] << 24) |
                                                (rxPacket->payload[3] << 16) |
                                                (rxPacket->payload[4] << 8)  |
                                                (rxPacket->payload[5]);

    latestRxPacket.dmSensorPacket.pressure =    (rxPacket->payload[6] << 24) |
                                                (rxPacket->payload[7] << 16) |
                                                (rxPacket->payload[8] << 8)  |
                                                (rxPacket->payload[9]);

    latestRxPacket.dmSensorPacket.humidity =    (rxPacket->payload[10] << 24) |
                                                (rxPacket->payload[11] << 16) |
                                                (rxPacket->payload[12] << 8)  |
                                                (rxPacket->payload[13]);

    latestRxPacket.dmSensorPacket.luminosity =  (rxPacket->payload[14] << 24) |
                                                (rxPacket->payload[15] << 16) |
                                                (rxPacket->payload[16] << 8)  |
                                                (rxPacket->payload[17]);

    latestRxPacket.dmSensorPacket.acceleration[0] = (rxPacket->payload[18] << 24) |
                                                    (rxPacket->payload[19] << 16) |
                                                    (rxPacket->payload[20] << 8)  |
                                                    (rxPacket->payload[21]);

    latestRxPacket.dmSensorPacket.acceleration[1] = (rxPacket->payload[22] << 24) |
                                                    (rxPacket->payload[23] << 16) |
                                                    (rxPacket->payload[24] << 8)  |
                                                    (rxPacket->payload[25]);
```

```
latestRxPacket.dmSensorPacket.acceleration[2] = (rxPacket->payload[26] << 24) |
                                                (rxPacket->payload[27] << 16) |
                                                (rxPacket->payload[28] << 8)  |
                                                (rxPacket->payload[29]);

latestRxPacket.dmSensorPacket.angularVelocity[0] = (rxPacket->payload[30] << 24) |
                                                   (rxPacket->payload[31] << 16) |
                                                   (rxPacket->payload[32] << 8)  |
                                                   (rxPacket->payload[33]);

latestRxPacket.dmSensorPacket.angularVelocity[1] = (rxPacket->payload[34] << 24) |
                                                   (rxPacket->payload[35] << 16) |
                                                   (rxPacket->payload[36] << 8)  |
                                                   (rxPacket->payload[37]);

latestRxPacket.dmSensorPacket.angularVelocity[2] = (rxPacket->payload[38] << 24) |
                                                   (rxPacket->payload[39] << 16) |
                                                   (rxPacket->payload[40] << 8)  |
                                                   (rxPacket->payload[41]);

latestRxPacket.dmSensorPacket.magneticField[0] = (rxPacket->payload[42] << 24) |
                                                 (rxPacket->payload[43] << 16) |
                                                 (rxPacket->payload[44] << 8)  |
                                                 (rxPacket->payload[45]);

latestRxPacket.dmSensorPacket.magneticField[1] = (rxPacket->payload[46] << 24) |
                                                 (rxPacket->payload[47] << 16) |
                                                 (rxPacket->payload[48] << 8)  |
                                                 (rxPacket->payload[49]);

latestRxPacket.dmSensorPacket.magneticField[2] = (rxPacket->payload[50] << 24) |
                                                 (rxPacket->payload[51] << 16) |
                                                 (rxPacket->payload[52] << 8)  |
                                                 (rxPacket->payload[53]);
```

```
latestRxPacket.dmSensorPacket.batt = (rxPacket->payload[54] << 8) | rxPacket->payload[55];
latestRxPacket.dmSensorPacket.time100MiliSec = (rxPacket->payload[56] << 24) |
                                               (rxPacket->payload[57] << 16) |
                                               (rxPacket->payload[58] << 8)  |
                                               (rxPacket->payload[59]);

/* Signal packet received */
Event_post(radioOperationEventHandle, RADIO_EVENT_VALID_PACKET_RECEIVED);
```

Modification du fichier « ConcentratorTask.c »

```
static void packetReceivedCallback(union ConcentratorPacket* packet, int8_t rssi)
{
    /* If we received an DualMode ADC sensor packet*/
    if(packet->header.packetType == RADIO_PACKET_TYPE_DM_SENSOR_PACKET)
    {

        /* Save the values */
        latestActiveSensorNode.address = packet->header.sourceAddress;

        latestActiveSensorNode.pressure          = (uint32_t) packet->dmSensorPacket.pressure;
        latestActiveSensorNode.acceleration[0]   = (uint16_t) packet->dmSensorPacket.acceleration[0];
        latestActiveSensorNode.acceleration[1]   = (uint16_t) packet->dmSensorPacket.acceleration[1];
        latestActiveSensorNode.acceleration[2]   = (uint16_t) packet->dmSensorPacket.acceleration[2];
        latestActiveSensorNode.magneticField[0]  = (int16_t) packet->dmSensorPacket.magneticField[0];
        latestActiveSensorNode.magneticField[1]  = (int16_t) packet->dmSensorPacket.magneticField[1];
        latestActiveSensorNode.magneticField[2]  = (int16_t) packet->dmSensorPacket.magneticField[2];
        latestActiveSensorNode.batt              = (int16_t) packet->dmSensorPacket.batt;
        latestActiveSensorNode.latestRssi = rssi;
        memcpy(&latestActiveSensorNode.temperature,        &packet->dmSensorPacket.temperature,
                                                              sizeof(packet->dmSensorPacket.temperature));
        memcpy(&latestActiveSensorNode.humidity,           &packet->dmSensorPacket.humidity,
                                                                 sizeof(packet->dmSensorPacket.humidity));
        memcpy(&latestActiveSensorNode.luminosity,         &packet->dmSensorPacket.luminosity,
                                                              sizeof(packet->dmSensorPacket.luminosity));
        memcpy(&latestActiveSensorNode.angularVelocity[0], &packet->dmSensorPacket.angularVelocity[0],
                                                              sizeof(packet->dmSensorPacket.angularVelocity[0]));
        memcpy(&latestActiveSensorNode.angularVelocity[1], &packet->dmSensorPacket.angularVelocity[1],
                                                              sizeof(packet->dmSensorPacket.angularVelocity[1]));
        memcpy(&latestActiveSensorNode.angularVelocity[2], &packet->dmSensorPacket.angularVelocity[2],
                                                              sizeof(packet->dmSensorPacket.angularVelocity[2]));

        Event_post(concentratorEventHandle, CONCENTRATOR_EVENT_UPDATE_SENSOR_VALUE);
    }
}
```

```
static void updateNode(struct SensorNode* node)
{
    uint8_t i;
    for (i = 0; i < CONCENTRATOR_MAX_NODES; i++)
    {
        if (knownSensorNodes[i].address == node->address)
        {
            knownSensorNodes[i].latestAdcValue = node->latestAdcValue;

            knownSensorNodes[i].temperature = node->temperature;
            knownSensorNodes[i].pressure = node->pressure;
            knownSensorNodes[i].humidity = node->humidity;
            knownSensorNodes[i].luminosity = node->luminosity;
            knownSensorNodes[i].acceleration[0] = node->acceleration[0];
            knownSensorNodes[i].acceleration[1] = node->acceleration[1];
            knownSensorNodes[i].acceleration[2] = node->acceleration[2];
            knownSensorNodes[i].angularVelocity[0] = node->angularVelocity[0];
            knownSensorNodes[i].angularVelocity[1] = node->angularVelocity[1];
            knownSensorNodes[i].angularVelocity[2] = node->angularVelocity[2];
            knownSensorNodes[i].magneticField[0] = node->magneticField[0];
            knownSensorNodes[i].magneticField[1] = node->magneticField[1];
            knownSensorNodes[i].magneticField[2] = node->magneticField[2];

            knownSensorNodes[i].latestRssi = node->latestRssi;
            knownSensorNodes[i].button = node->button;
           break;
        }
    }
}
```

```c
static void updateLcd(void)
{
    struct SensorNode* nodePointer = knownSensorNodes;
    uint8_t currentLcdLine;
    if(availableFwUpdateInProgress == false)
    {
        /* Start on the second line */
        currentLcdLine = 1;

        /* Write one line per node */
        while ((nodePointer < &knownSensorNodes[CONCENTRATOR_MAX_NODES]) &&
               (nodePointer->address != 0) &&
               (currentLcdLine < CONCENTRATOR_DISPLAY_LINES))
        {
            /* print to UART */
            Display_printf(hDisplaySerial, 0, 0,""
                    "{\"node\":0x%02x, "
                    "\"rssi\":%d, "
                    "\"battery\":%f, "
                    "\"temperature\":%f, "
                    "\"pressure\":%d, "
                    "\"humidity\":%f, "
                    "\"luminosity\":%f, "
                    "\"acceleration\":[%d, %d, %d],"
                    "\"angularvelocity\":[%f, %f, %f],"
                    "\"magneticfield\":[%d, %d, %d]}",
                    nodePointer->address,
                    nodePointer->latestRssi,
                    ((float)(nodePointer->batt))/256,
                    nodePointer->temperature,
                    nodePointer->pressure,
                    nodePointer->humidity,
                    nodePointer->luminosity,
                    nodePointer->acceleration[0],
                    nodePointer->acceleration[1],
                    nodePointer->acceleration[2],
                    nodePointer->angularVelocity[0],
                    nodePointer->angularVelocity[1],
                    nodePointer->angularVelocity[2],
                    nodePointer->magneticField[0],
                    nodePointer->magneticField[1],
                    nodePointer->magneticField[2]);

            nodePointer++;
            currentLcdLine++;
        }
```