

```

/*
 * Copyright 2014 UOC
 */

/**
 *
 * @file      main.cpp
 * @author    Albert Creixell
 * @version   v0.1
 * @date      Oct, 2014
 * @brief
 * @ingroup
 */

/*=====include =====*/

#include "string.h"

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

#include "openmote-cc2538.h"

#include "Callback.h"
#include "Serial.h"

/*=====define =====*/

#define PAYLOAD_LENGTH          ( 125 )
#define EUI48_LENGTH           ( 6 )
#define LIGHT_TASK_PRIORITY    ( tskIDLE_PRIORITY + 1 )
#define TEMPERATURE_TASK_PRIORITY ( tskIDLE_PRIORITY + 2 )

/*=====typedef =====*/

/*=====prototypes =====*/

extern "C" void vApplicationTickHook(void);
extern "C" void vApplicationIdleHook(void);
static void rtc_callback(void);
static PlainCallback rtcCallback(rtc_callback);

extern "C" TickType_t board_sleep(TickType_t xModifiableIdleTime);
extern "C" TickType_t board_wakeup(TickType_t xModifiableIdleTime);

static void iniciaranuraCord(void);
static void iniciaranuraEsc(void);
static void sincronitza(void);
static void sincronisme(void);
static void escriu(void);
static void llegeix(void);
static void incrementaASN(void);
static void incrementaRanura(void);
static void rxInit(void);
static void rxDone(void);

```

```

static void txInit(void);
static void txDone(void);

/*=====variables =====*/

static PlainCallback rxInitCallback(&rxInit);
static PlainCallback rxDoneCallback(&rxDone);
static PlainCallback txInitCallback(&txInit);
static PlainCallback txDoneCallback(&txDone);

static uint8_t radio_escriptura[PAYLOAD_LENGTH];
static uint8_t* radio_ptr = radio_escriptura;
static uint8_t radio_len = sizeof(radio_escriptura);
static RadioResult result;

static uint8_t radio_lectura[PAYLOAD_LENGTH];
static uint8_t* radio_ptr2 = radio_lectura;
static uint8_t radio_len2 = sizeof(radio_lectura);
static int8_t rssi;
static uint8_t lqi;
static uint8_t crc;

static uint8_t uart_buffer[PAYLOAD_LENGTH];
static uint8_t* uart_ptr = radio_escriptura;
static uint8_t uart_len = sizeof(radio_escriptura);

static Serial serial(uart);

/*=====TSCH =====*/
static uint8_t canal=26; //Variable per definir el canal radio a utilitzar
static uint8_t sincronitzat=0; //variable que defineix si l'esclau està sincronitzat
amb el coordinador
static bool assignat=false; //Variable que defineix si l'esclau té assignat o no una
ranura
static uint8_t ASN[4]; //Número ASN definit pel protocol TSCH
static uint8_t numranures=58; //Defineix el nombre de ranures que s'utilitzen
static uint8_t ordrecom[100][3]; //matriu on es guarden les assignacions de comunicacions
i el channel offset
static uint8_t cont_ranura=0; //Variable que controla en quina ranura es troba l'equip

static uint8_t estat=0; //Variable que defineix l'estat de la màquina d'estats
static uint8_t tipus=0; //Variable que defineix acció de ranura. 0-res, 1-escriu,
2-llegeix, 3-verifica sincronisme
static uint8_t ident=1; //Variable que defineix el número de Mote. Cada equip ha
de tenir una diferent
static bool coordinador=true; //Variable que indica si es o no coordinador
static uint32_t tic[7]; //Els tics s'utilitzen per guardar valors del rtc i calcular
temps
static int32_t retras; //Variable que marca el retràs o avanç en les comunicacions

static int32_t correccio; //Valor dels ticks a corregir en l'esclau
static bool registra;
static bool llegit=false;

/*=====public =====*/

int main (void)

```

```

{
    // Set the TPS62730 in bypass mode (Vin = 3.3V, Iq < 1 uA)
    //tps62730.setBypass();

    // Enable erasing the Flash with the user button
    //board.enableFlashErase();

    // Activa radio IEEE 802.15.4
    radio.setTxCallbacks(&txInitCallback, &txDoneCallback);
    radio.setRxCallbacks(&rxInitCallback, &rxDoneCallback);
    radio.enable();
    radio.enableInterrupts();

    //Inicialitza el vector de comunicacions
    ASN[0]=0;ASN[1]=0;ASN[2]=0;ASN[3]=0;ASN[4]=0;
    uint8_t contador;
    for(contador=0;contador<=numranures;contador++){
        ordrecom[contador][0]=0;ordrecom[contador][1]=0;ordrecom[contador][2]=0;ordrecom[
contador][3]=0;
    }
    // Activa el port sèrie del coordinador per poder veure dades
    uart.enable(UART_BAUDRATE, UART_CONFIG, UART_INT_MODE);
    serial.init();

    //Initialize Rtc
    rtc.init();
    rtc.setCallback(&rtcCallback);
    rtc.enableInterrupt();

    // Start Rtc
    rtc.start(327);

    // Enable interrupts
    board.enableInterrupts();

    // Activació I2C interface per adquirir les dades de temperatura, humitat i llum

    vTaskStartScheduler();

    // Forever
    /*while (true)
    {
        // Sleep
        board.setSleepMode(SleepMode::SleepMode_2);
        board.sleep();
    }*/
}

```

```

static void rtc_callback(void) //El callback de la interrupció del RTC s'utilitza per
controlar tot el procés
{
    if (coordinador==true){ //Codi que s'executa si es coordinador
        sincronitzat=1; //El coordinador és el master i la seva base de temps sempre és
la correcte
        if (estat==0){iniciaranuraCord();estat=1;} //Primer estat de la ranura, on s'inicialitza
que ha de fer
    }
}

```

```

    if (estat==1 && tipus==0){incrementaRanura();incrementaASN();estat=0;rtc.start(393
);} //Tipus=0 no fa res espera 12ms i inicia la ranura següent
    if (estat==1 && tipus==1){estat=2;} //Tipus=1 implica enviar informació
    if (estat==1 && tipus==2){estat=9;} //Tipus=2 implica llegir informació
    if (estat>1 && estat<9){escriu();} //estats d'escriptura
    if (estat>8 && estat<20){llegeix();} //estats de lectura
    }
    else{
        if (sincronitzat==1){ //Si es esclau i està sincronitzat executa aquest codi
            led_red.off();
            if (estat==0){iniciaranuraEsc();estat=1;}
            if (estat==1 && tipus==0){incrementaRanura();incrementaASN();estat=0;rtc.start
(393);} //Tipus=0 no fa res espera 12ms i inicia la ranura
            if (estat==1 && tipus==1){estat=2;}
            if (estat==1 && tipus==2){estat=9;}
            if (estat==1 && tipus==3){estat=18;}
            if (estat>1 && estat<9){escriu();}
            if (estat>8 && estat<18){llegeix();}
            if (estat>17 && estat<20){sincronisme();}
        }
        else{
            if (estat<20){estat=20;} //Estat de sincronització
            sincronitza();
        }
    }
}

static void iniciaranuraCord(void)
{
    uint64_t ASNC;
    ASNC=ASN[4]<<32;
    ASNC=ASNC+(ASN[3]<<24);
    ASNC=ASNC+(ASN[2]<<16);
    ASNC=ASNC+(ASN[1]<<8);
    ASNC=ASNC+(ASN[0]); //Converteix l'ASN en un número per calcular el mod
    if (cont_ranura==0){
        radio_escriptura[0]=255;
        radio_escriptura[1]=ASN[0];radio_escriptura[2]=ASN[1];radio_escriptura[3]=ASN[2];

        radio_escriptura[4]=ASN[3];radio_escriptura[5]=ASN[4];radio_escriptura[6]=numranures
;
        tipus=1;canal=26;radio.setChannel(canal);radio_len=7; //Envia l'ASN a tots els esclaus
perque sincronitzin

    }
    if (cont_ranura==1){
        tipus=2;canal=26;radio.setChannel(canal); //Ranura on escolta algun esclau i li assigna
ranura en el ACK
    }
    if (cont_ranura>=2){
        tipus=0;//Ranures on escriu, llegeix o no fa res segons les ranures assignades
        ASNC=ASNC+ordrecom[cont_ranura][2];//Suma al ASC el channel offset
        canal=(ASNC % 16)+11;
        if (ordrecom[cont_ranura][0]==ident){
            tipus=1; //Envia les dades de temperatura, humitat i llum
            radio_escriptura[0]=2;radio_escriptura[1]=55;
            radio_len=8;
        }
        if (ordrecom[cont_ranura][1]==ident){tipus=2;}
    }
}

```

```

        radio.setChannel(canal);
    }

}

static void iniciaranuraEsc(void)
{
    uint64_t ASNc;
    ASNc=ASN[4]<<32;
    ASNc=ASNc+(ASN[3]<<24);
    ASNc=ASNc+(ASN[2]<<16);
    ASNc=ASNc+(ASN[1]<<8);
    ASNc=ASNc+(ASN[0]);
    if (cont_ranura==0){
        tipus=3;canal=26;radio.setChannel(canal); //Cada inici verifica sincronisme i verifica
    l'ASN
    }
    if (cont_ranura==1){
        if (assignat==false){
            tipus=1;
            canal=26;radio.setChannel(canal); //Ranura demanarà si no enté ranura per rebre
        i enviar
            radio_escriptura[0]=10;
            radio_escriptura[1]=20;radio_escriptura[2]=ident;
            radio_escriptura[3]=0;radio_escriptura[4]=0;radio_escriptura[5]=0;
            radio_len=6;
        }
        else {tipus=0;}
    }
    if (cont_ranura>=2){
        tipus=0;//Ranures on escriu, llegeix o no fa res segons les ranures assignades
        ASNc=ASNc+ordrecom[cont_ranura][2];//Suma al ASC el channel offset
        canal=(ASNc % 16)+11;
        if (ordrecom[cont_ranura][0]==ident){
            tipus=1;//Envia les dades de temperatura, humitat i llum
            radio_escriptura[0]=1;radio_escriptura[1]=55;
            radio_len=8;
        }
        if (ordrecom[cont_ranura][1]==ident){tipus=2;}
        radio.setChannel(canal);
    }
}

static void sincronitza(void)
{
    if (estat==20){
        radio.setChannel(canal);
        led_red.on();
        assignat=false;
        radio.on();
        radio.receive();// Posa la radio a escoltar permanentment. En futur s'implementarà
    duty cycle
        estat=21;
    }
}

static void sincronisme(void)
{

```

```

    if (estat==19){sincronitzat=0;rtc.start(10);} //Si s'activa aquest estat implica
que no ha rebut l'ASN i activa la sincronització de l'esclau

    if (estat==18){
        tic[1]=rtc.read();
        radio.on();
        radio.receive();// Posa la radio a escoltar. Espera rebre l'ASN
        estat=19;
        rtc.start(383);
    }
}

static void escriu(void)
{
    if (estat==7){
        radio.off();
        correccio=0;
        if (coordinador==false){
            if (llegit==true && radio_lectura[0]==30 && radio_lectura[1]==60){

                correccio=radio_lectura[5]<<24;
                correccio=correccio+(radio_lectura[4]<<16);
                correccio=correccio+(radio_lectura[3]<<8);
                correccio=correccio+radio_lectura[2]; //Aqui detecta la correcció en ticks
a aplicar
            }
        }
        if (llegit==true && coordinador==false && cont_ranura==1 && assignat==false){

            uint8_t index=0;
            if (radio_lectura[0]==10 && radio_lectura[1]==25){
                index=radio_lectura[4];
                ordrecom[index][0]=ident;
                ordrecom[index][1]=1;
                ordrecom[index][2]=radio_lectura[5];
                ordrecom[index+1][0]=1;
                ordrecom[index+1][1]=ident;
                ordrecom[index+1][2]=radio_lectura[5];
                assignat=true; //Aquí si es esclau, no està assignat i ranura 1, llegeix
l'assignació de ranura de l'ACK
            }
        }
        incrementaRanura();incrementaASN();estat=0;
        tic[6]=rtc.read();
        int32_t total;
        tic[1]=393-(tic[6]-tic[5]);//Resta als 12ms el temps transcorregut per determinar
el temps a la següent ranura
        total=tic[1]-correccio; //Aplica la correcció marcada pel coordinador
        rtc.start(total);
    }
    if (estat==6){
        estat=7;
        tic[4]=rtc.read();
        rtc.start(10);//Es carrega 64 (2ms) al rtc per controlar el temps que triga en
enviar.
    }
    if (estat==5){
        llegit=false;
        tic[3]=rtc.read();

```

```

    radio.on();
    radio.receive();// Put the radio transceiver in receive mode
    estat=6;
    rtc.start(54);//Es carrega 64 (2ms) al rtc per controlar el temps que triga en
enviar.
}
if (estat==4){
    estat=5;
    tic[2]=rtc.read();
    rtc.start(64);//Es carrega 64 (2ms) al rtc per controlar el temps que triga en
enviar.
}
if (estat==3){
    tic[1]=rtc.read();
    radio.transmit();
    estat=4;rtc.start(163); //Es carrega 163 (5ms) al rtc per controlar el temps que
triga en enviar.
}
if (estat==2){
    tic[5]=rtc.read();
    radio.on();
    radio_ptr = radio_escriptura;
    result = radio.loadPacket(radio_ptr, radio_len);
    estat=3;
    rtc.start(98); //Carrega paquet a la radio i espera 3ms per enviar
}
}

static void llegeix(void)
{
    if (estat==13){
        incrementaRanura();incrementaASN();estat=0; //Finalitza lectura i si es esclau
aplica correcció de temps
        tic[6]=rtc.read();
        tic[5]=393-(tic[6]-tic[5]);
        rtc.start(tic[5]);
    }
    if (estat==12){
        if (llegit==true){
            radio.transmit();
        }
        tic[3]=rtc.read();estat=13;rtc.start(64); //Envia el ACK
    }
    if (estat==11){
        if (llegit==true){
            radio.on();
            if (coordinador==true){
                //Si coordinador, els bytes 2 i 3 indiquen el retràs detectat en el ACK

                radio_escriptura[0]=30;radio_escriptura[1]=60;
                radio_escriptura[2]=retras;radio_escriptura[3]=retras >> 8;
                radio_escriptura[4]=retras >> 16;radio_escriptura[5]=retras >> 24;
                if (cont_ranura==1){ //Si es la ranura 1 i coordinador envia en el ACK
la ranura a utilitzar
                    uint8_t contador;
                    registra=false;
                    contador=3;
                    if (radio_lectura[0]==10 && radio_lectura[1]==20){

```

```

        while(registra==false){
            if ((ordrecom[contador][0]==0 && ordrecom[contador][1]==0)
|| (ordrecom[contador][0]==radio_lectura[2])){
                //S'assigna ranura en parelles. Primer esclau envia i a
la següent ranura escolta al coordinador
                ordrecom[contador][0]=radio_lectura[2];
                ordrecom[contador][1]=ident;
                ordrecom[contador][2]=1; //Aquí indica el channel offset

                ordrecom[contador+1][0]=ident;
                ordrecom[contador+1][1]=radio_lectura[2];
                ordrecom[contador+1][2]=ordrecom[contador][2];
                registra=true;
                radio_escriptura[0]=10;radio_escriptura[1]=25;
                radio_escriptura[4]=contador;
                radio_escriptura[5]=ordrecom[contador][2];
            }
            else{
                if (contador>=numranures){registra=true;}
                else{contador=contador+2;}
            }
        }
    }
}

else{
    //Si no es coordinador en el ACK no s'indica correcció de retràs

    radio_escriptura[0]=30;radio_escriptura[1]=60;
    radio_escriptura[2]=0;radio_escriptura[3]=0;
    radio_escriptura[4]=0;radio_escriptura[5]=0;

}
radio_len=6;
radio_ptr = radio_escriptura;
    result = radio.loadPacket(radio_ptr, radio_len);
    estat=12;
    rtc.start(54); //Carrega paquet a la radio i espera 1ms per enviar

}
}
if (estat==10){estat=0;incrementaRanura();incrementaASN();radio.off();rtc.start(10
);}
if (estat==9){
    llegit=false;
    tic[5]=rtc.read();
    tic[1]=rtc.read();
    radio.on();
    radio.receive();// Put the radio transceiver in receive mode
    rtc.start(383); //Inicialitzem el rellotge per controlar el retràs. Idealment
les com comencen a tick 98
    estat=10;
}
}
}

static void incrementaASN(void)

```



```

{
    if (ASN[0]==255){ASN[0]=0;ASN[1]++;}
    else {ASN[0]++;}

    if (ASN[1]==255){ASN[1]=0;ASN[2]++;}
    if (ASN[2]==255){ASN[2]=0;ASN[3]++;}
    if (ASN[3]==255){ASN[3]=0;ASN[4]++;}
    if (ASN[4]==255){ASN[4]=0;}
}

static void incrementaRanura(void)
{
    cont_ranura++;
    if (cont_ranura>numranures){
        cont_ranura=0;
    }
}

static void rxInit(void)
{
    // Codi que s'executa quan comença a rebre la radio. Serveix per detectar retards
    if (estat==10){tic[2]=rtc.read();retras=(tic[2]-tic[1])-98;}
    if (estat==19){tic[2]=rtc.read();}
    if (estat==21){tic[2]=rtc.read();}
    led_green.on();
}

static void rxDone(void)
{
    //Codi que s'executa al acabar de rebre, serveix per llegir i fer canvis d'estat.
    led_green.off();
    llegit=true;
    if (estat==6){
        tic[4]=rtc.read();
        radio_ptr2 = radio_lectura;
        radio_len2 = sizeof(radio_lectura);
        result = radio.getPacket(radio_ptr2, &radio_len2, &rssi, &lqi, &crc);
        estat=7;rtc.start(10);}

    if (estat==10){estat=11;
        radio_ptr2 = radio_lectura;
        radio_len2 = sizeof(radio_lectura);
        result = radio.getPacket(radio_ptr2, &radio_len2, &rssi, &lqi, &crc);

        radio.off();
        tic[2]=rtc.read();
        rtc.start(10);}

    if (estat==19){
        incrementaRanura();incrementaASN();estat=0;
        radio.off();
        int32_t retras2; //Variable que marca el retràs o avanç en les comunicacions

        retras2=98-(tic[2]-tic[1]);
        radio_lectura[0]=30;radio_lectura[1]=60;
        radio_lectura[2]=retras2;radio_lectura[3]=retras2 >> 8;
        radio_lectura[4]=retras2 >> 16;radio_lectura[5]=retras2 >> 24;
        uart_ptr = uart_buffer;
    }
}

```

```

        memcpy(uart_ptr, radio_ptr2, 6);
        uart_ptr += radio_len2;
        uart_len = radio_len2;
    uart.writeByte(uart_buffer, uart_len);

    tic[3]=rtc.read();
    tic[1]=(393-(tic[2]-tic[1])-(tic[3]-tic[2]));
    rtc.start(tic[1]);}

    if (estat==21){
    tic[3]=rtc.read();
    radio_ptr2 = radio_lectura;
        radio_len2 = sizeof(radio_lectura);
        result = radio.getPacket(radio_ptr2, &radio_len2, &rssi, &lqi, &crc);
    if (radio_lectura[0]==255){
        ASN[0]=radio_lectura[1];ASN[1]=radio_lectura[2];
        ASN[2]=radio_lectura[3];ASN[3]=radio_lectura[4];
        ASN[4]=radio_lectura[5];
        numranures=radio_lectura[6]; //L'esclau rep i adapta el número de ranures
        sincronitzat=1;cont_ranura=0;
        incrementaRanura();incrementaASN();estat=0;
        radio.off();
        rtc.start(393-98-(tic[3]-tic[2])); //Aproximadament la pròxima ranura serà a 8ms.
        L'ajust fi es farà en la ranura 1 amb l'ACK
    }
    }
}

static void txInit(void)
{
    // Rutina que indica quan comença a enviar. Nomes s'utilitza per encendre un led
    led_yellow.on();
}

static void txDone(void)
{
    //Rutina que indica que ha acabat d'enviar. Serveix per fer canvis d'estat
    led_yellow.off();
    if (estat==4){
        radio.off();
        estat=5;
        tic[2]=rtc.read();
        rtc.start(64);}
    if (estat==12){
        radio.off();
        tic[4]=rtc.read();
        estat=13;rtc.start(1);}
}

```