

```

//-----//
//
//          Actuador lumínic
//
//-----//

// Llibreria PlatformIO
#include <Arduino.h>

//Llibreries per LoRa
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

//Llibreria RTC
#include <driver/rtc_io.h>

//Configuració RTC i Deep Sleep

RTC_DATA_ATTR lmic_t RTC_LMIC;
bool GOTO_DEEPSLEEP = false;

// APPEUI Little-endian
static const u1_t PROGMEM APPEUI[8]={ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// DEVEUI Little-endian
static const u1_t PROGMEM DEVEUI[8]={ 0xE2, 0x00, 0x05, 0xD0, 0x7E, 0xD5,
0xB3, 0x70 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// APPKEY Big-endian
static const u1_t PROGMEM APPKEY[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
void os_getDevKey (u1_t* buf) { memcpy_P(buf, APPKEY, 16);}

// Data types variables

```

```

RTC_DATA_ATTR uint8_t payload[4];
static osjob_t sendjob;

// Duty Cycle enviament TX Uplink
const unsigned TX_INTERVAL = 1;

// Variable LLUM
#define LED_Vermell GPIO_NUM_33

// Pin mapping EZSBC i RFM95W
const lmic_pinmap lmic_pins = {
    .nss = 5,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 4,
    .dio = {2, 21, 22},
};

// Conversió dades
void printHex2(unsigned v) {
    v &= 0xff;
    if (v < 16)
        Serial.print('0');
    Serial.print(v, HEX);
}

// Enviament dades
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {

        Serial.println(payload[0]);
        Serial.println(payload[1]);
        Serial.println(payload[2]);
        Serial.println(payload[3]);
    }
}

```

```

        // Preparem les dades per la següent transmissió TX.
        LMIC_setTxData2(1, payload, sizeof(payload), 0);
        Serial.println(F("Packet queued"));
    }
}

```

// Event que respon segons l'estat de les comunicacions

```

void LoraWANPrintLMICOpmode(void)
{
    Serial.print(F("LMIC.opmode: "));
    if (LMIC.opmode & OP_NONE)
    {
        Serial.print(F("OP_NONE "));
    }
    if (LMIC.opmode & OP_SCAN)
    {
        Serial.print(F("OP_SCAN "));
    }
    if (LMIC.opmode & OP_TRACK)
    {
        Serial.print(F("OP_TRACK "));
    }
    if (LMIC.opmode & OP_JOINING)
    {
        Serial.print(F("OP_JOINING "));
    }
    if (LMIC.opmode & OP_TXDATA)
    {
        Serial.print(F("OP_TXDATA "));
    }
    if (LMIC.opmode & OP_POLL)
    {
        Serial.print(F("OP_POLL "));
    }
    if (LMIC.opmode & OP_REJOIN)
    {
        Serial.print(F("OP_REJOIN "));
    }
}

```

```
}  
  
if (LMIC.opmode & OP_SHUTDOWN)  
{  
    Serial.print(F("OP_SHUTDOWN "));  
}  
  
if (LMIC.opmode & OP_TXRXPEND)  
{  
    Serial.print(F("OP_TXRXPEND "));  
}  
  
if (LMIC.opmode & OP_RNDTX)  
{  
    Serial.print(F("OP_RNDTX "));  
}  
  
if (LMIC.opmode & OP_PINGINI)  
{  
    Serial.print(F("OP_PINGINI "));  
}  
  
if (LMIC.opmode & OP_PINGABLE)  
{  
    Serial.print(F("OP_PINGABLE "));  
}  
  
if (LMIC.opmode & OP_NEXTCHNL)  
{  
    Serial.print(F("OP_NEXTCHNL "));  
}  
  
if (LMIC.opmode & OP_LINKDEAD)  
{  
    Serial.print(F("OP_LINKDEAD "));  
}  
  
if (LMIC.opmode & OP_LINKDEAD)  
{  
    Serial.print(F("OP_LINKDEAD "));  
}  
  
if (LMIC.opmode & OP_TESTMODE)  
{  
    Serial.print(F("OP_TESTMODE "));  
}
```

```

    if (LMIC.opmode & OP_UNJOIN)
    {
        Serial.print(F("OP_UNJOIN "));
    }
}

//Funció que avisa de l'estat LoRa
void LoraWANDebug(lmic_t lmic_check)
{
    Serial.println("");
    Serial.println("");

    LoraWANPrintLMICOpmode();
    Serial.println("");

    Serial.print(F("LMIC.seqnoUp = "));
    Serial.println(lmic_check.seqnoUp);

    Serial.print(F("LMIC.globalDutyRate = "));
    Serial.print(lmic_check.globalDutyRate);
    Serial.print(F(" osTicks, "));
    Serial.print(osticks2ms(lmic_check.globalDutyRate) / 1000);
    Serial.println(F(" sec"));

    Serial.print(F("LMIC.globalDutyAvail = "));
    Serial.print(lmic_check.globalDutyAvail);
    Serial.print(F(" osTicks, "));
    Serial.print(osticks2ms(lmic_check.globalDutyAvail) / 1000);
    Serial.println(F(" sec"));

    Serial.print(F("LMICbandplan_nextTx = "));
    Serial.print(LMICbandplan_nextTx(os_getTime()));
    Serial.print(F(" osTicks, "));
    Serial.print(osticks2ms(LMICbandplan_nextTx(os_getTime())) / 1000);
    Serial.println(F(" sec"));

    Serial.print(F("os_getTime = "));

```

```

Serial.print(os_getTime());
Serial.print(F(" osTicks, "));
Serial.print(osticks2ms(os_getTime()) / 1000);
Serial.println(F(" sec"));

Serial.print(F("LMIC.txend = "));
Serial.println(lmic_check.txend);
Serial.print(F("LMIC.txChnl = "));
Serial.println(lmic_check.txChnl);

Serial.println(F("Band \tavail \t\tavail_sec\tlastchnl \ttxcap"));
for (u1_t bi = 0; bi < MAX_BANDS; bi++)
{
    Serial.print(bi);
    Serial.print("\t");
    Serial.print(lmic_check.bands[bi].avail);
    Serial.print("\t\t");
    Serial.print(osticks2ms(lmic_check.bands[bi].avail) / 1000);
    Serial.print("\t\t");
    Serial.print(lmic_check.bands[bi].lastchnl);
    Serial.print("\t\t");
    Serial.println(lmic_check.bands[bi].txcap);
}
Serial.println("");
Serial.println("");
}

// Funció Per saber el temps d'execució
void PrintRuntime()
{
    long seconds = millis() / 1000;
    Serial.print("Runtime: ");
    Serial.print(seconds);
    Serial.println(" seconds");
}

// Funció per guardar les dades correctes

```

```

void SaveLMICToRTC(int deepsleep_sec)
{
    Serial.println(F("Guardant LMIC to RTC"));
    RTC_LMIC = LMIC;

    // Reset DutyCycles
    unsigned long now = millis();

    // EU Like Bands
    #if defined(CFG_LMIC_EU_like)
        Serial.println(F("Reset CFG_LMIC_EU_like band avail"));
        for (int i = 0; i < MAX_BANDS; i++)
        {
            ostime_t correctedAvail = RTC_LMIC.bands[i].avail - ((now / 1000.0
+ deepsleep_sec) * OSTICKS_PER_SEC);
            if (correctedAvail < 0)
            {
                correctedAvail = 0;
            }
            RTC_LMIC.bands[i].avail = correctedAvail;
        }

        RTC_LMIC.globalDutyAvail = RTC_LMIC.globalDutyAvail - ((now / 1000.0 +
deepsleep_sec) * OSTICKS_PER_SEC);
        if (RTC_LMIC.globalDutyAvail < 0)
        {
            RTC_LMIC.globalDutyAvail = 0;
        }
    #else
        Serial.println(F("No DutyCycle recalculation function!"));
    #endif
}

// Funció per carregar la informació després de despertar
void LoadLMICFromRTC()
{
    Serial.println(F("Carregar LMIC des de RTC"));
    LMIC = RTC_LMIC;
}

```

```

}

// Funció per dormir el dispositiu
void GoDeepSleep()
{
    // DEEPSLEEP
    Serial.println(F("Go DeepSleep"));
    // Mirem quan ha durat la retransmissió
    PrintRuntime();
    //Apaguem el Serial
    Serial.flush();
    // Posem a dormir
    ESP.deepSleep(TX_INTERVAL * 1000000);
}

// Funció que respon segons l'estat de les comunicacions
void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT"));
            break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND"));
            break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED"));
            break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED"));
            break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING"));
            break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));

```



```

{
    u4_t netid = 0;
    devaddr_t devaddr = 0;
    u1_t nwkKey[16];
    u1_t artKey[16];
    LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey);
    Serial.print("netid: ");
    Serial.println(netid, DEC);
    Serial.print("devaddr: ");
    Serial.println(devaddr, HEX);
    Serial.print("AppSKey: ");
    for (size_t i=0; i<sizeof(artKey); ++i) {
        if (i != 0)
            Serial.print("-");
        printHex2(artKey[i]);
    }
    Serial.println("");
    Serial.print("NwkSKey: ");
    for (size_t i=0; i<sizeof(nwkKey); ++i) {
        if (i != 0)
            Serial.print("-");
        printHex2(nwkKey[i]);
    }
    Serial.println();
}

LMIC_setLinkCheckMode(0);
break;

///< case EV_RFU1:
///<     Serial.println(F("EV_RFU1"));
///<     break;
///<
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));

```

```

        break;

    case EV_TXCOMPLETE:

        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));

        if (LMIC.txrxFlags & TXRX_ACK)

            Serial.println(F("Received ack"));

        if (LMIC.dataLen) {

            Serial.print(F("Received "));

            Serial.print(LMIC.dataLen);

            Serial.println(F(" bytes of payload"));

            //----- Recepció Dades Downlink -----

            if (LMIC.dataLen == 1) {

                uint8_t result = LMIC.frame[LMIC.dataBeg + 0];

                if (result == 0) {

                    Serial.println("RESULT 0");

                    rtc_gpio_hold_dis(LED_Vermell); //Deshabilitar el PIN RTC

                    rtc_gpio_set_level(LED_Vermell, LOW); //set high/low

                    rtc_gpio_hold_en(LED_Vermell); //Habilitar el pin RTC

                    payload[0] = 0;

                }

                if (result == 1) {

                    Serial.println("RESULT 1");

                    rtc_gpio_hold_dis(LED_Vermell); //Deshabilitar el PIN RTC

                    rtc_gpio_set_level(LED_Vermell, HIGH); //set high/low

                    rtc_gpio_hold_en(LED_Vermell); //Habilitar el pin RTC

                    payload[0] = 1;

                }

            }

            Serial.println();

        }

        // Activem la variable per la següent iteració anar a dormir

        GOTO_DEEPSLEEP = true;

        break;

```

```

case EV_LOST_TSYNC:
    Serial.println(F("EV_LOST_TSYNC"));
    break;

case EV_RESET:
    Serial.println(F("EV_RESET"));
    break;

case EV_RXCOMPLETE:
    // Dades rebudes
    Serial.println(F("EV_RXCOMPLETE"));
    break;

case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
    break;

case EV_LINK_ALIVE:
    Serial.println(F("EV_LINK_ALIVE"));
    break;

//|| case EV_SCAN_FOUND:
//||     Serial.println(F("EV_SCAN_FOUND"));
//||     break;

case EV_TXSTART:
    Serial.println(F("EV_TXSTART"));
    break;

case EV_TXCANCELED:
    Serial.println(F("EV_TXCANCELED"));
    break;

case EV_RXSTART:
    //No escriure res o el timing no anirà bé
    break;

case EV_JOIN_TXCOMPLETE:
    Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
    break;

default:
    Serial.print(F("Unknown event: "));
    Serial.println((unsigned) ev);

```

```

        break;
    }
}

void setup() {
    setCpuFrequencyMhz(10); // Millora de 80 mA a 30 mA
    delay(10);

    Serial.begin(115200);
    Serial.println(F("Starting"));

    //Configurar el pin LED a la memòria RTC
    rtc_gpio_init(LED_Vermell); //Inicialitzem el PIN GPIO RTC
    rtc_gpio_set_direction(LED_Vermell, RTC_GPIO_MODE_OUTPUT_ONLY); //Definim
    el mode RTC del PIN

    // LMIC init
    os_init();
    // Reset MAC
    LMIC_reset();

    //Revisem si cal volcar la informació guardada dins la memòria RTC
    if (RTC_LMIC.seqnoUp != 0)
    {
        LoadLMICFromRTC();
    }

    //S'executa per la realització de proves i saber el seu estat
    LoraWANDebug(LMIC);

    // Start OTTA
    do_send(&sendjob);
}

void loop() {

    static unsigned long lastPrintTime = 0;

```

```

//Funció LoRa
os_runloop_once();

//Funció Intermitent que revisa si cal anar a dormir
const bool timeCriticalJobs =
os_queryTimeCriticalJobs(ms2osticksRound((TX_INTERVAL * 1000)));

if (!timeCriticalJobs && GOTO_DEEPSLEEP == true && !(LMIC.opmode &
OP_TXRXPEND))
{
    Serial.print(F("Anem a dormir "));
    LoraWANPrintLMICOpmode();
    SaveLMICToRTC(TX_INTERVAL);
    GoDeepSleep();
}
else if (lastPrintTime + 2000 < millis())
{
    Serial.print(F("No pot dormir "));
    Serial.print(F("TimeCriticalJobs: "));
    Serial.print(timeCriticalJobs);
    Serial.print(" ");

    LoraWANPrintLMICOpmode();
    PrintRuntime();
    lastPrintTime = millis();
}
}

```