Multi-platform IoT Architectures

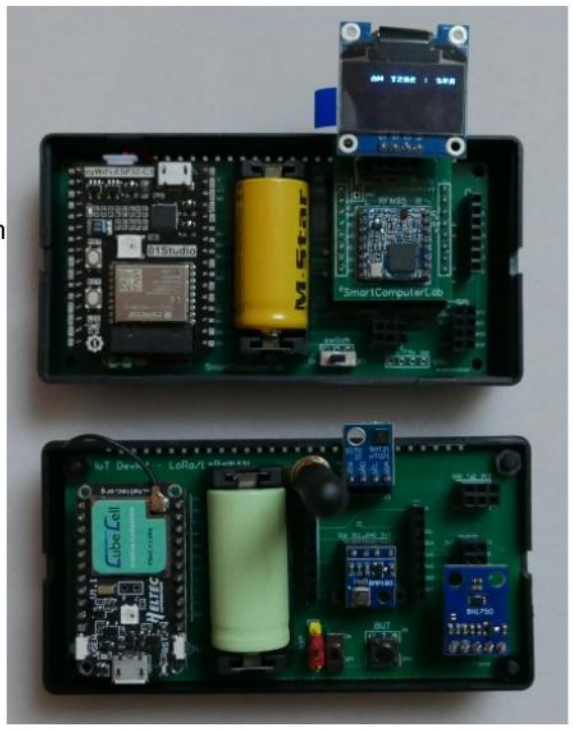
RISC-V (µPython) and ARM (C/C+)

Long Range and Low Power – LoRa communication

Table of Contents

0. Introduction

In the following IoT labs we are going to experiment with two novel IoT architectures and Long Range communication based on sx1276 and sx1762 LoRa modems.
The processing part of these IoT nodes are of two types: RISC-V (ESP32-C3) and ARM – Cortex-M0.



- **PyCOM-V** : **RISC**-V based SoC has sufficient processing power and memory space to operate with microPython interpreter on the top of FreeRTOS. Note the extension board with LoRa **sx1276 modem**

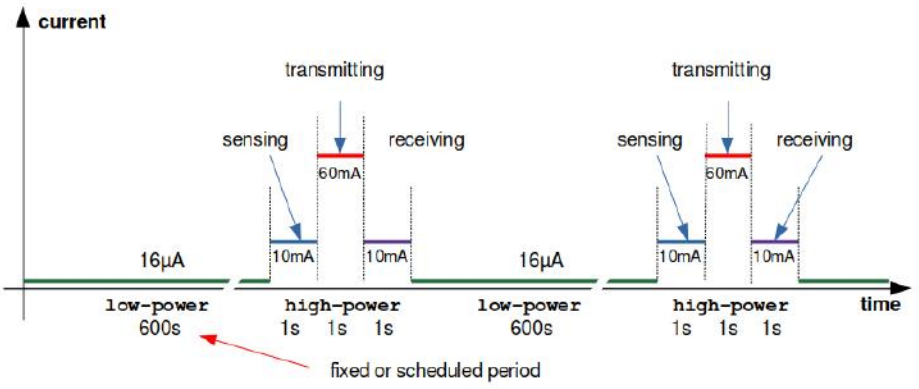
- **LowPowCOM**: **ARM+LoRa** SoM (ASR6501) is optimized for low power operation and efficient LoRa communication based on **sx1262 modem**.

We start with the introduction of ARM+LoRa SoM incorporated into CubeCell boards. These in turn are integrated in our Low-Power IoT DevKit.

The main objective of the labs is to show how to implement a very low power consumption applications with the terminal nodes integrating small LiPo batteries and miniature solar panels. The proposed IoT configurations should operate over very long time periods (months-years).
Low power operation of the terminal nodes is based on several mechanism that allows us to put the MCU in **low_power** mode, to put the LoRa radio in **sleep mode**, and to **cut the current** flow use by the sensors. Having all of these three conditions fulfilled we can downscale the overall current to **10-20 µA** with 3.3V voltage.

This dormant state can be interrupted by the timeout signal starting the wake period with high_power consumption period. The **high_power** consumption period may be decomposed in several phases including sensing phase, transmission phase, reception phase with response data storage. The energy consumption during the **sensing phase** depends on the characteristics of the sensor(s) including the sensing time and the current.

The following diagram shows the operational line of a terminal.



The energy consumption during the transmission phase depends on the transmission power and the duration of the transmission operation. These in turn depend on the size of the data packet and the LoRa radio parameters used for the transmission such as signal bandwidth, spreading factor and code-rate.

The duration of the **low_power** period may be fixed or provided dynamically by the gateway node through the time-out parameter sent in the response packet. This mechanism called **Target Wake Time** (**TWT**) is very efficient and also may be used to build the scheduled operation of a bigger number of terminal nodes.
TWT based scheduling allows us to avoid the collisions between the transmitted packets.

The response also may carry the **delta parameter** informing the Terminal node about the variation of the sensor values to be taken into account for the transmission. For example we can imagine 3 different delta values for (1 for 0.1%, 10 for 1%, and 100 for 10%) the variation of the sensing values.

Figure 0.1 Low_power and **high_power** periods of operational cycle

0.1 Hardware Architecture - main board (CubeCell)

Heltec CubeCell board (HTCC-AB01) uses the ASR6501 module. The ASR6501 is SoM (System-on-Module) that combines an ARM Cortex-M0+ 32 bits 48MHz MCU (with 16kB SRAM and 128kB flash) together with a Semtech SX1262 LoRa transceiver in a single package. The CubeCell products have an integrated **LoRaWAN stack** based on Semtech's LoRaMac-node.

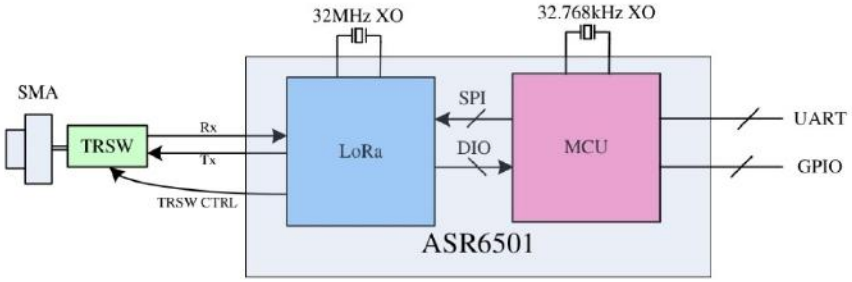


Figure 0.2 ASR6501 architectural scheme with MCU and Lora modem connected via SPI bus

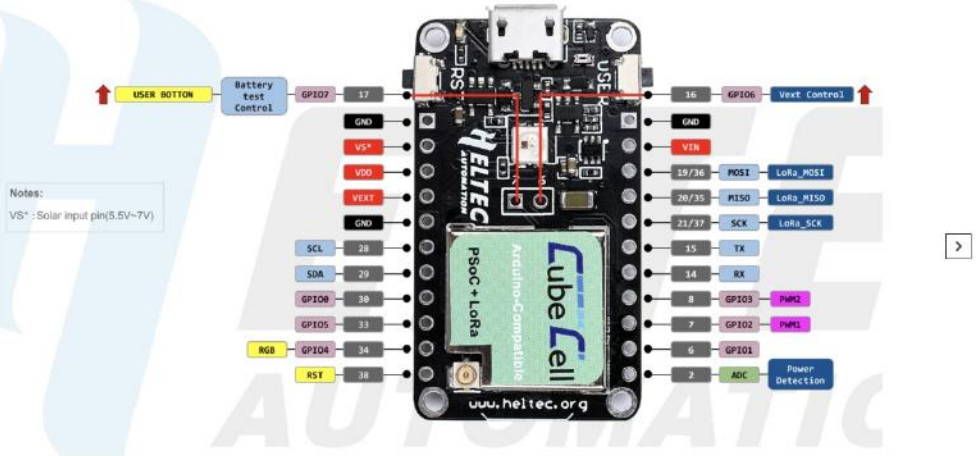


Figure 0.3 CubeCell board and its pinout

The implementation of the IoT terminals is mainly driven by the very low power consumption and the communication capabilities including LoRaWAN protocol.
The following are the power consumption characteristics:

- Supply current in Sleep mode: 10-20 µA
- Supply current in Receiver mode: 10-20 mA
- Supply current in Transmitter mode: 80-100 mA

Imagine a terminal sending the data frames of 16 bytes once every 10 minutes (10*60*1000 ms). With the spreading factor of 8 and the bandwidth of 125kHz, the calculated airtime is: 123.4 ms (~ 125 ms)
It means that the average current consumption in mA is:

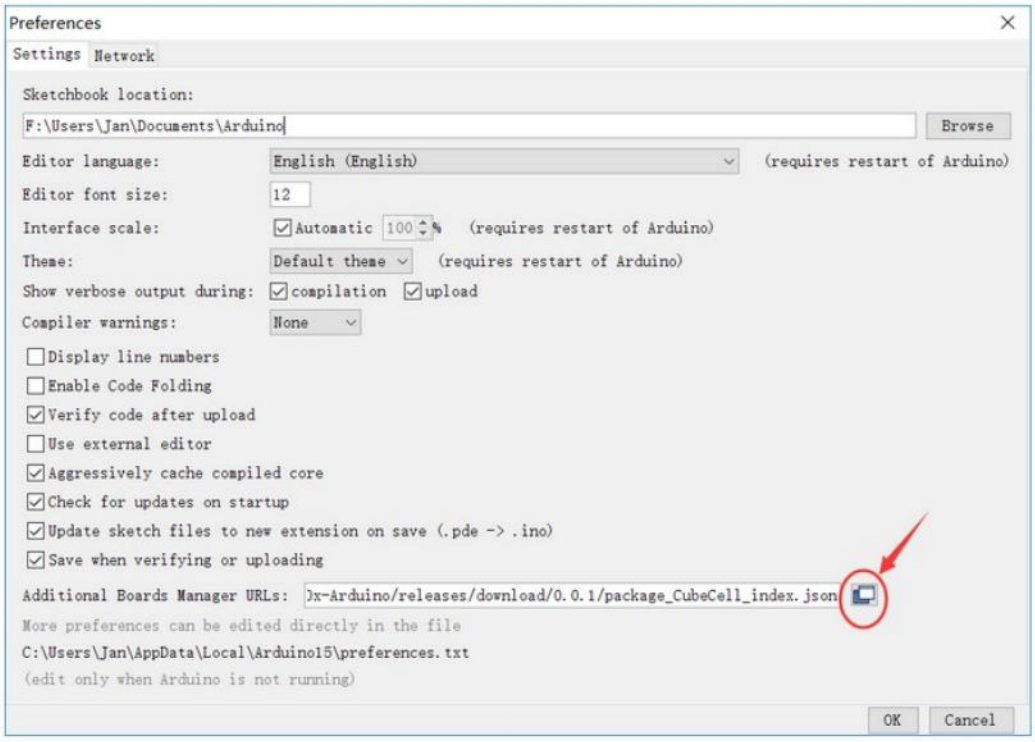
(125*100 000+10*60*1000)/60*1000=(125*100+10*60)/60=(12500+60)/60=210 µA=0.21 mA

A small battery with the capacity of 1000 mAh has the possibility of power supply during:
1000/0.21=4761 hours that is almost 200 days.
Even if we divide this result by 2 we still have 3 months of power supply.

CellCube integrates the solar panel interface (6-7V) for small panels with 100mv to 1W power output. These solar components provide much longer operation periods.
The CubeCell products support development with the Arduino framework. Sketches are uploaded via the serial port. The development boards have a USB port with USB-to-serial so sketches can be easily uploaded via USB. Uploading sketches to the sensor capsules requires a special adapter from Heltec.
In June 2020 supported for PlatformIO was added. Heltec uses a custom CubeCell bootloader for ASR650x. Serial number and a license that enables Arduino support are stored in flash memory.

Software installation with Arduino board manager

Open Arduino IDE, and click **`File→Peferences→Settings`**



https://github.com/HelTecAutomation/CubeCell-Arduino/releases/download/V1.3.0/package_CubeCell_index.json

Principal link for software resources:

`https://github.com/leroyle/ASR650x-Arduino`

Lab 1 – CubeCell platform: main board and sensors

In this initial labs we are going to test a number of essential **features** of the **HTCC-AB01** board. The knowledge of these features is necessary for the understanding of the following laboratories.

We are inviting you to test these examples before starting the work on Long Range communication.

1.1 Board chip identifier

```
#include "Arduino.h"
void setup() {
  Serial.begin(9600);
  delay(100);
}

void loop() {
Serial.println("in the loop");
  uint64_t chipID=getID();delay(100);
  Serial.println();Serial.println();
  Serial.printf("ChipID:%04X%08X\r\n",(uint32_t)(chipID>>32),(uint32_t)chipID);
delay(1000);
}
```

1.2 Integrated RGB LED

The following is the basic code to test the integrated **RGB LED.**

```
#include "CubeCell_NeoPixel.h"
CubeCell_NeoPixel pixels(1, RGB, NEO_GRB + NEO_KHZ800);

void setup() {
  pinMode(Vext,OUTPUT);
  digitalWrite(Vext,LOW); //SET POWER
  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
  pixels.clear(); // Set all pixel colors to 'off'
}
uint8_t i=0;

void loop() {
    pixels.setPixelColor(0, pixels.Color(i, 0, 0));
    pixels.show();   // Send the updated pixel colors to the hardware.
    delay(200); // Pause before next pass through loop
    pixels.setPixelColor(0, pixels.Color(0, i, 0));
    pixels.show();   // Send the updated pixel colors to the hardware.
    delay(200); // Pause before next pass through loop
    pixels.setPixelColor(0, pixels.Color(0, 0, i));
    pixels.show();   // Send the updated pixel colors to the hardware.
    delay(200); // Pause before next pass through loop
    i+=10;
}
```

1.3 Integrated USR button

The following is the basic code to test the integrated **USR button.**

```
#include "Arduino.h"
#define USR GPIO7  // user button
uint32_t cnt = 0;

void cntIncrease()
{
  cnt++;
  Serial.println(cnt);
}

void setup() {
  Serial.begin(9600);
  PINMODE_INPUT_PULLUP(USR);
  attachInterrupt(USR,cntIncrease,FALLING);
}

void loop() {}
```

1.4 User flash memory.

The chip has 1K user flash:

- the size of user flash row is 256;
- user flash row 0-2 can be edited;
- user flash row 3 is reserved, must not be edited;

```
#include "Arduino.h"
#define ROW 0
#define ROW_OFFSET 100

//CY_FLASH_SIZEOF_ROW is 256 , CY_SFLASH_USERBASE is 0x0ffff400
#define addr CY_SFLASH_USERBASE+CY_FLASH_SIZEOF_ROW*ROW + ROW_OFFSET

uint8_t data1[512];
uint8_t data2[512];
uint8_t data3;
uint8_t data4;
void setup() {
  Serial.begin(9600);delay(100);
  Serial.println();Serial.println();
  for(int i=0;i<512;i++){
    data1[i]=(uint8_t)i;
  }
  //write data1 to flash at addr
  FLASH_update(addr,data1,sizeof(data1));
  //read flash at addr to data2
  FLASH_read_at(addr,data2,sizeof(data2));
  uint16_t error=0;
  for(int i=0;i<512;i++){
    if(data1[i]!=data2[i])
    {
      Serial.printf("error:data1[%d] %d , data2[%d] %d \r\n",i,data1[i],i,data2[i]);
      error++;
    }
  }
  Serial.printf("error:%d\r\n",error);
  //read a byte at addr to data4
  FLASH_read_at(addr,&data4,1);
  Serial.printf("data4:%d\r\n",data4);
  data3=100;
  //write a byte at addr
  FLASH_update(addr,&data3,1);
  //read a byte at addr to data4
  FLASH_read_at(addr,&data4,1);
  Serial.printf("data4:%d\r\n",data4);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

The terminal printout:

```
error:0
data4:0
data4:100
```

1.5 Sleep modes: timer and user interruption

1.5.1 Timer mode

```
#include "Arduino.h"
#include "LoRa_APP.h"
#define timetillsleep 5000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;

void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Radio.Sleep( );  // LoRa modem sleep mode
  TimerInit( &sleep, onSleep );
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  // put your main code here, to run repeatedly:
}
```

1.5.2 User interrupt mode

```
#include "Arduino.h"
#include "LoRa_APP.h"

#define INT_GPIO USER_KEY
#define timetillsleep 5000
static TimerEvent_t sleep;
uint8_t lowpower=1;

void onSleep()
{
  Serial.printf("Going into lowpower mode. Press user key to wake up\r\n");
  delay(5);
  lowpower=1;
}
void onWakeUp()
{
  delay(10);
  if(digitalRead(INT_GPIO) == 0)
  {
          Serial.printf("Woke up by GPIO, %d ms later into lowpower mode.\r\n",timetillsleep);
          lowpower=0;
          //timetillsleep ms later into lowpower mode;
          TimerSetValue( &sleep, timetillsleep );
          TimerStart( &sleep );
  }
}
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(INT_GPIO,INPUT);
  attachInterrupt(INT_GPIO,onWakeUp,FALLING);
  TimerInit( &sleep, onSleep );
  Serial.printf("Going into lowpower mode. Press user key to wake up\r\n");
  delay(5);
}

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  // put your main code here, to run repeatedly:
}
```

1.6 System time

```
TimerSysTime_t sysTimeCurrent;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  /*typedef struct TimerSysTime_s
  *{
  *  uint32_t Seconds;
  *  int16_t SubSeconds;
  *}TimerSysTime_t;
  */
  sysTimeCurrent = TimerGetSysTime( );
  Serial.printf("sys time:%u.%d\r\n",(unsigned int)sysTimeCurrent.Seconds,
sysTimeCurrent.SubSeconds);
  TimerSysTime_t newSysTime ;
  newSysTime.Seconds = 1000;
  newSysTime.SubSeconds = 50;
  TimerSetSysTime( newSysTime );
  sysTimeCurrent = TimerGetSysTime( );
  Serial.printf("sys time:%u.%d\r\n",(unsigned int)sysTimeCurrent.Seconds,
sysTimeCurrent.SubSeconds);
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(1000);
  sysTimeCurrent = TimerGetSysTime( );
  Serial.printf("sys time:%u.%d\r\n",(unsigned int)sysTimeCurrent.Seconds,
sysTimeCurrent.SubSeconds);
}
```

1.7 WatchDog timer

Watchdog timers are commonly found in embedded systems and **IoT equipment** where humans cannot easily access the equipment or would be unable to react to faults in a timely manner. In such systems, the computer cannot depend on a human to invoke a reboot if it hangs ; it must be self-reliant.

For example, remote embedded systems such as space probes are not physically accessible to human operators; these could become permanently disabled if they were unable to autonomously recover from faults. In robots and other automated machines, a fault in the control computer could cause equipment damage or injuries before a human could react, even if the computer is easily accessed. A watchdog timer is usually employed in cases like these.
Watchdog timers are also used to monitor and **limit software execution time** on a normally functioning computer.

For example, a watchdog timer may be used to limit the CPU time available to the code and thus prevent some types of denial-of-service attacks. In real time system a watchdog timer may be used to monitor a time-critical task to ensure it completes within its maximum allotted time and, if it fails to do so, to terminate the task and report the failure.

```
#include "Arduino.h"
#include "innerWdt.h"
// For asr650x, the max feed time is 2.8 seconds.
#define MAX_FEEDTIME 2000 // default is 2800 ms
bool autoFeed = false;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println();
  Serial.println("Start");
  /* Enable the WDT.
  * autoFeed = false: do not auto feed wdt.
  * autoFeed = true : it auto feed the wdt in every watchdog interrupt.
  */
  innerWdtEnable(autoFeed);
}

int feedCnt = 0;

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("running");
  delay(MAX_FEEDTIME - 100);

  if(autoFeed == false)
  {
    //feed the wdt
    if(feedCnt < 3)
    {
      Serial.println("feed wdt");
      feedInnerWdt();
      feedCnt++;
    }
    else
    {
      Serial.println("stop feed wdt");
    }
  }
}
```

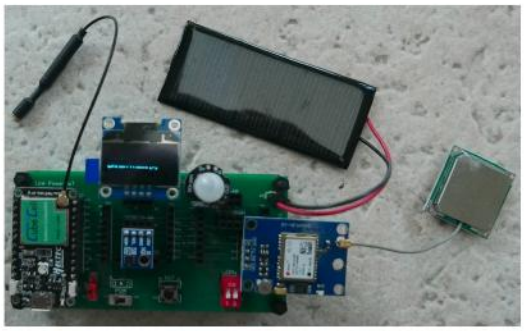
To do

- Install the required software
- Test the above examples

1.8 External components and low power operation

In this lab we are building complete nodes using external components such as sensors, GPS modems, displays and relays. All these components are connected to the main board via I2C, UART bus or via simple logic lines. We start **SSD1306 OLED** display connected via **I2C** bus.

1.8.1 SSD1306 OLED



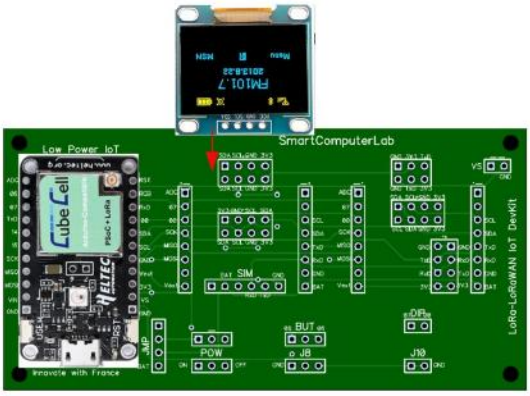


Fig 1.1 IoT DevKit with OLED screen connected via I2C bus

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include "HT_SSD1306Wire.h"
#include "Wire.h"
SSD1306Wire  display(0x3c, 500000, SDA, SCL, GEOMETRY_128_64, -1);

void displayOLED(char *line1, char *line2, char *line3)
{

    display.init();
    display.flipScreenVertically();display.clear();
    display.drawString(20, 50, "SmartComputerLab" );
    display.drawString(0, 0,  line1 );
    display.drawString(0, 15, line2);
    display.drawString(0, 30, line3);
    display.display();
}

void setup() {
    Serial.begin(9600);
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);
    delay(100);
    Wire.begin(29,28);
    display.init();
    display.flipScreenVertically();
}

int c1=0,c2=0,c3=0;

void loop()
{
char l1[32],l2[32],l3[32];
sprintf(l1,"Count1=%d",c1);sprintf(l2,"Count2=%d",c2);
sprintf(l3,"Count3=%d",c3);
c1+=1;c2+=2;c3+=3;
displayOLED(l1,l2,l3);
delay(2000);
}
```

1.8.2 SHT21 sensor module (I2C)

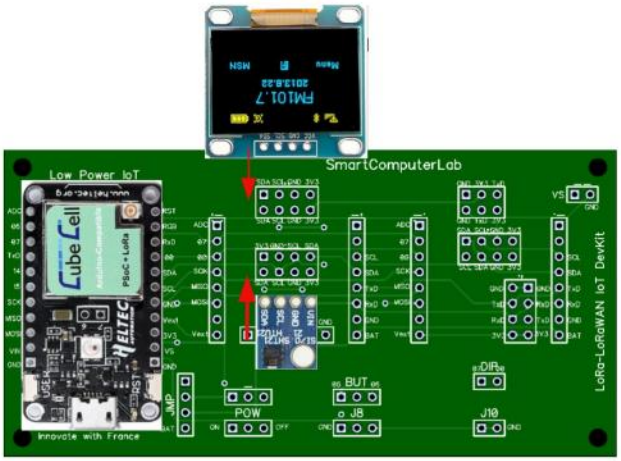


Fig 1.2 IoT DevKit with OLED screen and SHT21 sensor connected via I2C bus

An example sketch that reads the sensor and prints the relative humidity to the serial port

```
#include <Wire.h>
#include "SHT21.h"

SHT21 SHT21;

float t,h;

void setup()
{
  pinMode(Vext,OUTPUT);
  digitalWrite(Vext,LOW); //3V3 voltage output activated - Vext ON
  Wire.begin(29,28);
  SHT21.begin();
  Serial.begin(9600);
}

void loop()
{
  char buff[32];
  t=SHT21.getTemperature();
  h=SHT21.getHumidity();
  Serial.print("Humidity(%RH): ");
  Serial.print(h);
  Serial.print("     Temperature(C): ");
  Serial.println(t);
  dt=(int)((t-(int)t)*100.0);  dh=(int)((h-(int)h)*100.0);
  sprintf(buff,"T:%d.%d, H:%d.%d\n",(int)t,dt,(int)h,dh);
  Serial.println(buff);
  delay(1000);
}
```

Reading SHT21 sensor and display on OLED

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include "HT_SSD1306Wire.h"
#include <Wire.h>
#include "SHT21.h"

SHT21 SHT21;

SSD1306Wire  display(0x3c, 500000, SDA, SCL, GEOMETRY_128_64, -1);  // addr , freq , i2c group ,
resolution , rst

void displayOLED(char *line1, char *line2, char *line3)
{
    display.init();
    display.flipScreenVertically();display.clear();
    display.drawString(20, 50, "SmartComputerLab" );
    display.drawString(0, 0,  line1 );
    display.drawString(0, 15, line2);
    display.drawString(0, 30, line3);
    display.display();
}

float t,h;
int dt,dh;
char buff[32];
```

```
void getSHT21()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW); delay(100);

    SHT21.begin();
    t=SHT21.getTemperature();
    h=SHT21.getHumidity();
    Serial.print("Humidity(%RH): ");
    Serial.print(h);
    Serial.print("     Temperature(C): ");
    Serial.println(t);
    dt=(int)((t-(int)t)*100.0);  dh=(int)((h-(int)h)*100.0);
    sprintf(buff,"T:%d.%d, H:%d.%d\n",(int)t,dt,(int)h,dh);
    Serial.println(buff); delay(100);displayOLED(buff,NULL,NULL);delay(2000);

    digitalWrite(Vext, HIGH); delay(100);
}



void setup()
{
  Serial.begin(9600);
  while(!Serial);Serial.println();
  Wire.begin(29,28);
  display.init();
  display.flipScreenVertically();
}

void loop()
{
  getSHT21();
  delay(6000);
}
```

1.8.3 BH1750 – luminosity sensor


```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

void setup()
{
  Serial.begin(9600);
  pinMode(Vext,OUTPUT);
  digitalWrite(Vext,LOW); //3V3 voltage output activated - Vext ON
  Wire.begin(29,28);
  lightMeter.begin();
  Serial.println(F("BH1750 Test begin"));
}

void loop() {
  float lux = lightMeter.readLightLevel();
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

To do

- Test the above examples
- Add the display on the OLED screen to show the luminosity value
- use **`digitalWrite(Vext,LOW);`** and **`digitalWrite(Vext,HIGH );`** to set on and off the power line **`Vext`** in the **`loop()`** function

1.8.4 `SoftwareSerial` – GPS - NEO-6MV2

This sample code demonstrates the normal use of a **`TinyGPS++`** (TinyGPSPlus) object. It requires the use of **`SoftwareSerial`**, and assumes that you have a 9600-baud serial GPS device hooked up on pins **`GPIO3`** - (**`rx`**) and **`GPIO5`** – (**`tx`**).

1.8.4.1 Simple – direct UART stream

```
#include <Arduino.h>
#include <softSerial.h>
softSerial softwareSerial(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);  // GPIO5 (33) , GPIO3 (8)

void setup()
{
pinMode(Vext, OUTPUT);
digitalWrite(Vext, LOW);
delay(500);
Serial.begin(9600);
softwareSerial.begin(9600);
delay(1000);
Serial.println("Normal serial init");
}
char *ptr, gmt[12],clarg[12], clong[12];

void loop()
{
  if(softwareSerial.available())
  {
  char serialbuffer[256] = {0};
  int i = 0;
  while (softwareSerial.available() && i<256)
    {
    serialbuffer[i] = (char)softwareSerial.read();
    i++;
    }
  serialbuffer[i] = '\0';
  if(serialbuffer[0])
    {
    Serial.println(serialbuffer);
    ptr=strstr(serialbuffer,"RMC,");
    strncpy(gmt,ptr+4,6); Serial.print("GMT:");Serial.println(gmt);
    ptr=strstr(serialbuffer,",A,");
    strncpy(clarg,ptr+3,10); Serial.print("Larg:");Serial.println(clarg);
    ptr=strstr(serialbuffer,",N,");
    strncpy(clong,ptr+3,10); Serial.print("Long:");Serial.println(clong);
    Serial.println();
    }
  }
delay(200);
}
```

1.8.4.2 Simple – direct UART stream and OLED display

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include "HT_SSD1306Wire.h"
#include "Wire.h"
#include <softSerial.h>
// The serial connection to the GPS device
softSerial softwareSerial(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);  // GPIO5 (33) , GPIO3 (8)
SSD1306Wire  display(0x3c, 500000, SDA, SCL, GEOMETRY_128_64, -1);  // addr,freq,i2c group ,
resolution , rst

void displayOLED(char *line1, char *line2, char *line3)
{
    display.init();
    display.flipScreenVertically();display.clear();
    display.drawString(20, 50, "SmartComputerLab" );
    display.drawString(0, 0,  line1 );
    display.drawString(0, 15, line2);
    display.drawString(0, 30, line3);
    display.display();
}
```

```
void setup()
{

pinMode(Vext, OUTPUT);
digitalWrite(Vext, LOW);
delay(500);
Serial.begin(9600);
Wire.begin(29,28);
softwareSerial.begin(9600);
delay(1000);
Serial.println("Normal serial init");
}
char *ptr, gmt[12],clarg[12], clong[12],dgmt[24],dclarg[24], dclong[24];
void loop()
{
if(softwareSerial.available())
{
char serialbuffer[256] = {0};
int i = 0;
while (softwareSerial.available() && i<256)
{
serialbuffer[i] = (char)softwareSerial.read();
i++;
}
serialbuffer[i] = '\0';
if(serialbuffer[0])
{
//Serial.print("Received data from software Serial:");
Serial.println(serialbuffer);
ptr=strstr(serialbuffer,"RMC,");
strncpy(gmt,ptr+4,6); Serial.print("GMT:");Serial.println(gmt);
ptr=strstr(serialbuffer,",A,");
strncpy(clarg,ptr+3,10); Serial.print("Larg:");Serial.println(clarg);
ptr=strstr(serialbuffer,",N,");
strncpy(clong,ptr+3,10); Serial.print("Long:");Serial.println(clong);
Serial.println();
sprintf(dgmt,"GMT:%s",gmt);sprintf(dclarg,"LAT:%s",clarg);sprintf(dclong,"LNG:%s (W)",clong);
displayOLED(dgmt, dclarg, dclong);
}
}
delay(600);
}
```

1.8.4.3 Simple – direct UART stream and OLED display with `low_power` mode

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include "HT_SSD1306Wire.h"
#include "Wire.h"
#include <softSerial.h>
softSerial softwareSerial(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);  // GPIO5 (33) , GPIO3 (8)

SSD1306Wire  display(0x3c, 500000, SDA, SCL, GEOMETRY_128_64, -1);
// addr , freq , i2c group , resolution , rst

#define timetillsleep 5000
#define timetillwakeup 10000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1, highpower=0;

void displayOLED(char *line1, char *line2, char *line3)
{
    display.init();
    display.flipScreenVertically();display.clear();
    display.drawString(20, 50, "SmartComputerLab" );
    display.drawString(0, 0,  line1 );
    display.drawString(0, 15, line2);
    display.drawString(0, 30, line3);
    display.display();
}
void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
```

```
  lowpower=1;highpower=0;
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;highpower=1;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup()
{
pinMode(Vext, OUTPUT);
digitalWrite(Vext, LOW);
delay(500);
Serial.begin(9600);
Wire.begin(29,28);
softwareSerial.begin(9600);
delay(1000);
Serial.println("Normal serial init");
TimerInit( &sleep, onSleep );
TimerInit( &wakeUp, onWakeUp );
onSleep();
}

char *ptr, gmt[12],clarg[12], clong[12],dgmt[24],dclarg[24], dclong[24];

void loop()
{
  if(lowpower){
    lowPowerHandler();
  }
  if(highpower)
    {
    if(softwareSerial.available())
      {
      char serialbuffer[256] = {0};
      int i = 0;
      while (softwareSerial.available() && i<256)
        {
        serialbuffer[i] = (char)softwareSerial.read();
        i++;
        }
      serialbuffer[i] = '\0';
      if(serialbuffer[0])
        {
        //Serial.print("Received data from software Serial:");
        Serial.println(serialbuffer);
        ptr=strstr(serialbuffer,"RMC,");
        strncpy(gmt,ptr+4,6); Serial.print("GMT:");Serial.println(gmt);
        ptr=strstr(serialbuffer,",A,");
        strncpy(clarg,ptr+3,10); Serial.print("Larg:");Serial.println(clarg);
        ptr=strstr(serialbuffer,",N,");
        strncpy(clong,ptr+3,10); Serial.print("Long:");Serial.println(clong);
        Serial.println();
        sprintf(dgmt,"GMT:%s",gmt);sprintf(dclarg,"LAT:%s",clarg);sprintf(dclong,"LNG:%s
(W)",clong);
        displayOLED(dgmt, dclarg, dclong);
        }
      }
    }
  //digitalWrite(Vext, HIGH);
  delay(4000);
}
```

To do

- Test the above examples of **GPS** receivers
- Add the display on the OLED screen to show the value of time, longitude, and latitude for the second example code

1.8.5 Low power operation period with `lowPowerHandler()`

In this lab we have already introduced low power technique to read and display the data on sensors and OLED screen with cutting off the power line during the idle period.
Now we add the **low_power** periods for the operation of the processor (MCU) itself.

To start lets us study the following example.

The first part of the code adds the necessary libraries and the declaration of the sensor (SHT21).

```
#include "Arduino.h"
#include <Wire.h>
#include "SHT21.h"
SHT21 SHT21;
```

The second fragment contains the definition and the declaration of the constants and variables used to configure the events required for low and high power operation (periods).

```
#define timetillsleep 5000
#define timetillwakeup 10000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1, highpower=0;
```

`timetosleep` and **`timetillwakeup`** define the periods of high and low power operation. In this case the board will be set for low power period of 10 secs and high power period of 5 secs.
The corresponding timer events are **`sleep`** for the high-to-low power period and **`wakeUp`** for the low-to-high power period.
`lowpower` and **`highpower`** variables keep the actual state of the operation.

The next section of the code defines the **ISR functions** activated by the time on **`sleep`** and on **`wakeUp`** events. They are:

```
void onSleep()
{
  Serial.printf("Go to lowpower mode,%d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;highpower=0;  //
  //timetillwakeup ms later wake up;
  TimerSetValue(&wakeUp,timetillwakeup);
  TimerStart(&wakeUp);
}

void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;highpower=1;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue(&sleep,timetillsleep);
  TimerStart(&sleep);
}
```

The setup section of the code initializes the pointers to **`onSleep`** and on **`WakeUp`** functions. The radio modem (LoRa) is set into sleep mode by **`Radio.Sleep()`** .
The execution of the program starts by **`onSleep()`** function in order to set the application into initial low power mode.

```
void setup()
{
  Serial.begin(9600);
  Radio.Sleep( );
  TimerInit( &sleep, onSleep );
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}
```

The main **`loop()`** of the program oscillates between the low-power and high-power modes.
During the high-power period the **`getSHT21()`** function is called – **only once**.

```
void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  if(highpower)
     { getSHT21();highpower=0; }
```

The last part of the code contains the getSHT21() function.
The execution of this function starts by the initialization of the power (3V3) line – **`Vext`** and by the activation of the I2C bus by **`Wire.begin(29,28)`** ; then we activate the sensor to read the temperature and humidity data.

After the data readings the I2C bus is deactivated – **`Wire.end()`** and the power line is cutoff - **`digitalWrite(Vext, HIGH)`**.

```
float t,h;
int dt,dh;
char buff[32];

void getSHT21()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);
    delay(50);
    Wire.begin(29,28);
    SHT21.begin();
    t=SHT21.getTemperature();
    h=SHT21.getHumidity();
    Serial.print("Humidity(%RH): ");
    Serial.print(h);
    Serial.print("     Temperature(C): ");
    Serial.println(t);
    dt=(int)((t-(int)t)*100.0);  dh=(int)((h-(int)h)*100.0);
    sprintf(buff,"T:%d.%d, H:%d.%d\n",(int)t,dt,(int)h,dh);
    Serial.println(buff);
    Wire.end();
    digitalWrite(Vext, HIGH);
}
```

The complete code is given below:

```
#include "Arduino.h"
#include <Wire.h>
#include "SHT21.h"
SHT21 SHT21;

#define timetillsleep 5000
#define timetillwakeup 10000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1, highpower=0;

float t,h;
int dt,dh;
char buff[32];

void getSHT21()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);
    delay(50);
    Wire.begin(29,28);
    SHT21.begin();

    t=SHT21.getTemperature();
    h=SHT21.getHumidity();
    Serial.print("Humidity(%RH): ");
    Serial.print(h);
    Serial.print("     Temperature(C): ");
    Serial.println(t);
    dt=(int)((t-(int)t)*100.0);  dh=(int)((h-(int)h)*100.0);
```

```
    sprintf(buff,"T:%d.%d, H:%d.%d\n",(int)t,dt,(int)h,dh);
    Serial.println(buff);
    Wire.end();
    digitalWrite(Vext, HIGH);
}

void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;highpower=0;
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;highpower=1;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Radio.Sleep( );
  TimerInit( &sleep, onSleep );
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}

void loop() {
  if(lowpower){
    lowPowerHandler();
  }

  if(highpower)
     { getSHT21();highpower=0; }
```

The resulting current consumption values from the integrated LiPo battery are:

- **11 mA** for **high power** operations
- **25 µA** for **low power** operations

To do

1. Apply the same low/high power operational mode for other sensors.
2. Try to use more than one sensor and the OLED display.

Lab 2 - LoRa modem - basics and low power operation

The following examples show how to use the integrated LoRa modem (SX1262) in basic mode (**pure LoRa**). These examples take into account low power requirements including implemented through MCU **low_power** state **radio sleep** and the **power cutoff** for the external components (sensors, modems, displays, ..)

2.1 LoRa send with battery state value

In this first example of LoRa based communication we send simple LoRa packets using the functions provided in **`LoRaWan_APP.h`** library. The packets contain the value of the battery state provided in **`mV`**.

LoRa radio channel is configured with several parameters including:

```
#define RF_FREQUENCY                                868000000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm - max 21 dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       8         // [SF7..SF12]
#define LORA_CODINGRATE                             4         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
```

The **`LoRaWan_APP.h`** library includes :

```
Radio.Send( (uint8_t *)txPacket, strlen(txPacket) );
```

to send a packet

and

```
RadioEvents.RxDone = OnRxDone;
Radio.Init( &RadioEvents );

..
  void OnRxDone(uint8_t *payload,uint16_t size,int16_t rssi,int8_t snr ) { }
```

to receive Lora packets.

```
Radio.Sleep();
```

Puts the LoRa modem into deep sleep state. The activated modem requires at least **10 mA** to stay awake.

Our first code example starts with the configuration of the radio parameters including: the frequency, the signal bandwidth, the spreading factor, the coding rate, etc.
At the receiving side we need to use the same parameters.

The integrated RGB LED needs about **5 mA** to be active , if you may deactivate the LED with:

```
turnOffRGB();
```

To activate the LED you may use:

```
turnOnRGB(COLOR_SEND,0); - red
turnOnRGB(COLOR_RECEIVED,0); - green
```

In order to keep the power consumption during the passive periods as low as possible you have to use these 3 functions:

- `lowPowerHandler();`
- `Radio.Sleep();`
- `turnOffRGB();`

2.1.1 Sending data packet

This first example of LoRa sender does not implement the low power features, the code just sends the value of the battery voltage.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

char txPacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );

typedef enum
{
    LOWPOWER,  ReadVoltage,  TX   // 3 states (1,2,3)
} States_t;

States_t state;
bool sleepMode = false;
int16_t rssi,rxSize;
uint16_t voltage;

void setup()
{
    Serial.begin(9600);
    voltage = 0;
    rssi=0;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );

    state=ReadVoltage;
}

void loop()
{
  switch(state)
  {
    case TX:
    {
```

```
      sprintf(txPacket,"%s","ADC_battery (mV): ");
      sprintf(txPacket+strlen(txPacket),"%d",voltage);
      if(voltage<(uint16_t)3680)turnOnRGB(COLOR_SEND,0);
      else turnOnRGB(COLOR_RECEIVED,200);
      Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txPacket, strlen(txPacket));
      Radio.Send( (uint8_t *)txPacket, strlen(txPacket) );
      state=LOWPOWER;
      break;
    }
    case LOWPOWER:
    {
      lowPowerHandler();
      delay(100);
      turnOffRGB();
      delay(2000);  //LowPower time
      state = ReadVoltage;
      break;
    }
    case ReadVoltage:
    {
      pinMode(VBAT_ADC_CTL,OUTPUT);
      digitalWrite(VBAT_ADC_CTL,LOW);
      voltage=analogRead(ADC)+550; // adjusted value
      pinMode(VBAT_ADC_CTL, INPUT);
      state = TX;
      break;
    }
     default:
          break;
  }
  Radio.IrqProcess();
}

void OnTxDone( void )
{
  Serial.print("TX done!");
  turnOnRGB(0,0);
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.print("TX Timeout......");
    state=ReadVoltage;
    Serial.print(state);
}
```

2.1.2 Sending data packet with low power management

In the following example we introduce the element of power management. During the passive period we downgrade the activity of the MCU with **`lowPowerHandler()`** , we suspend the activity of the LoRa modem with **`Radio.Sleep()`**, and we deactive integrated LED with t**`urnOffRGB()`**.

These preparations allow us to reduce the power consumption (current) to about **20 µA** during the passive periods defined as - **`timetillwakeup 15000`**.

During the active period defined by - **`timetillsleep 1000`** , the voltage capture and the LoRa packet transmission, power consumption goes up to **60 mA or more** depending on the transmission power **`TX_OUTPUT_POWER`** parameter.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#define timetillsleep 1000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
```

```
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here


char txPacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );

typedef enum {LOWPOWER,ReadVoltage,TX} States_t;
States_t state;

bool sleepMode = false;
int16_t rssi,rxSize;
uint16_t voltage;

void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;       turnOffRGB(); //Radio.Sleep();
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup() {
    Serial.begin(9600);
    voltage = 0;
    rssi=0;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
    state=ReadVoltage;

    Radio.Sleep( );  // LoRa modem sleep mode
    TimerInit( &sleep, onSleep );
    TimerInit( &wakeUp, onWakeUp );
    onSleep();
}

void loop()
{
if(lowpower)
  {
    lowPowerHandler();
  }
else
  {
```

```
  switch(state)
  {
    case TX:
    {
      sprintf(txPacket,"%s","ADC_battery (mV): ");
      sprintf(txPacket+strlen(txPacket),"%d",voltage);
      if(voltage<(uint16_t)3680)turnOnRGB(COLOR_SEND,0);
      else turnOnRGB(COLOR_RECEIVED,0);
      Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txPacket, strlen(txPacket));
      Radio.Send( (uint8_t *)txPacket, strlen(txPacket) );
      state=LOWPOWER;delay(100);
      break;
    }
    case LOWPOWER:
    {
      Serial.println("going to low power mode");
      delay(100);
      turnOffRGB();
      delay(100);  //LowPower time
      state = ReadVoltage;
      break;
    }
    case ReadVoltage:
    {
      Serial.println("reading battery voltage");
      pinMode(VBAT_ADC_CTL,OUTPUT);
      digitalWrite(VBAT_ADC_CTL,LOW);
      voltage=analogRead(ADC)*2;
      pinMode(VBAT_ADC_CTL, INPUT);
      state = TX;
      break;
    }
     default:
          break;
  }
  Radio.IrqProcess();
 }
}


void OnTxDone( void )
{
  Serial.println("TX done!");
  turnOffRGB();Radio.Sleep( );
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.println("TX Timeout......");
    state=ReadVoltage;
    Serial.println(state);
}
```

To do

1. Analyze the above codes
2. Test the examples

2.2 LoRa - receiving the packets with battery state

At the reception side we use the same headers as in the sender code to keep the identical parameters. The receiver is always active but it stays in sleep mode awaiting the new packet that is signaled by the associated interruption captured by **`Radio.IrqProcess( );`**

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // Hz
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];

static RadioEvents_t RadioEvents;
int16_t txNumber;
int16_t rssi,rxSize;

void setup() {
    Serial.begin(9600);

    txNumber=0;
    rssi=0;

    RadioEvents.RxDone = OnRxDone;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );

  Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                                 LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                                 LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                                 0, true, 0, 0, LORA_IQ_INVERSION_ON, true );
   turnOnRGB(COLOR_SEND,0); //change rgb color
   Serial.println("into RX mode");
   }

void loop()
{
  Radio.Rx( 0 );
  delay(500);
  Radio.IrqProcess( );
}

void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    rssi=rssi;
    rxSize=size;
    memcpy(rxpacket, payload, size );
    rxpacket[size]='\0';
    turnOnRGB(COLOR_RECEIVED,0);
    Radio.Sleep( );
    Serial.printf("\r\nreceived packet \"%s\" with rssi %d , length %d\r\n",rxpacket,rssi,rxSize);

}
```

The IDE terminal output (fragment):

```
..
received packet "ADC_battery (mV): 3886" with rssi -41 , length 22
received packet "ADC_battery (mV): 3886" with rssi -40 , length 22
received packet "ADC_battery (mV): 3886" with rssi -41 , length 22
received packet "ADC_battery (mV): 3886" with rssi -39 , length 22
received packet "ADC_battery (mV): 3886" with rssi -40 , length 22
received packet "ADC_battery (mV): 3886" with rssi -41 , length 22
received packet "ADC_battery (mV): 3886" with rssi -39 , length 22
received packet "ADC_battery (mV): 3886" with rssi -40 , length 22
..
```

2.3 Sending LoRa packets with sensor data

In this section we are going to add an external sensor connected via I2C bus. This sensor (SHT21) will provide us with two data values: temperature and humidity. In this context you can use any other sensor to provide luminosity, TVOC/eCO2 , atmospheric pressure, etc.

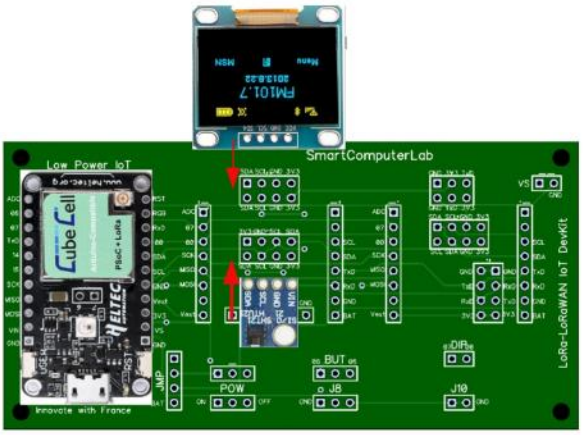


Fig 2.1 IoT DevKit for Low Power IoT design with main board and SHT21 sensor.

The following **power consumption diagram** for the terminal node shows the current values required for different period and phases of the operational cycle.

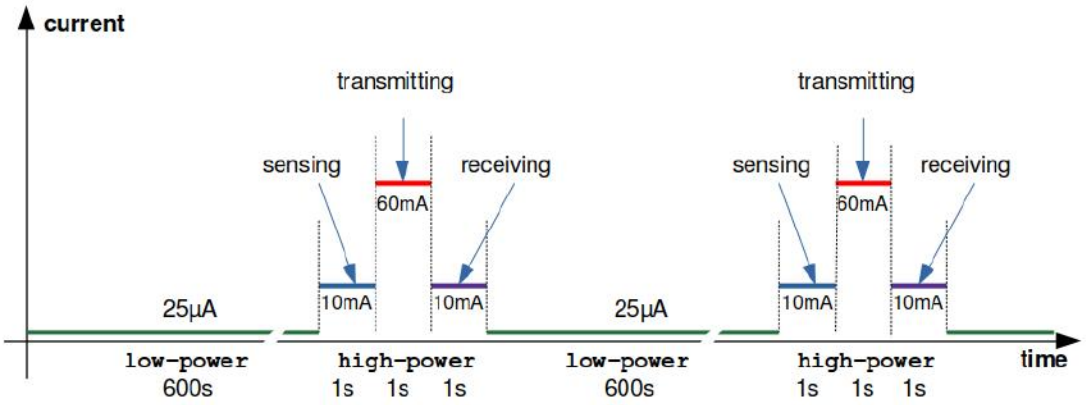


Fig 2.2 Power consumption diagram (current for 3.3V) with low and high power periods and high-power operational phases.

SF	Chirps / Symbol	SNR	Airtime [a]	Bitrate
7	128	−7.5	56.5 ms	5469 bps
8	256	−10	−103 ms	3125 bps
9	512	−12.5	185.3 ms	1758 bps
10	1024	−15	371 ms	977 bps
11	2048	−17.5	741 ms	537 bps
12	4096	−20	1318.9 ms	293 bps

[a] 20 bytes per packet and Code Rate = 4/5.

Fig 2.3 Transmission – airtime during high power period for 20-byte payload and 4/5 coderate

Example of power consumption evaluation in µA for SF=8, CR=4/8, and 16 byte payload:

- airtime = 103*(16/20)*(8/5) ms = 103*128/100 = 132 ms (less than 200 ms)
- transmission current for 14 dBm (25 mW radio) we need about 40 mA with 3.3V

Let us calculate the average current consumption for :

- 600 sec **low_power** period
- 2.2 sec **high_power** period with 3 phases

(600*25 + 1*10000+0.2*60000+1*10000)/603 = 157,545605307 => 80 µA

For a battery with the capacity of 1000 mAh.

1000*1000/80 =6410,25 => 12500 hours or 12500/750 => 16 months

That is more than a year of operation.

In general the transmitting current depends on TX power set to :

- for 14 dBm (25 mW radio) we need about 40 mA (3.3V)
- for 17 dBm (50 mW radio) we need about 60 mA (3.3V)
- for 20 dBm (100 mW radio) we need about 100 mA (3.3V)

The following figure shows the transmission time as a function of payload size and the spreading factor value.

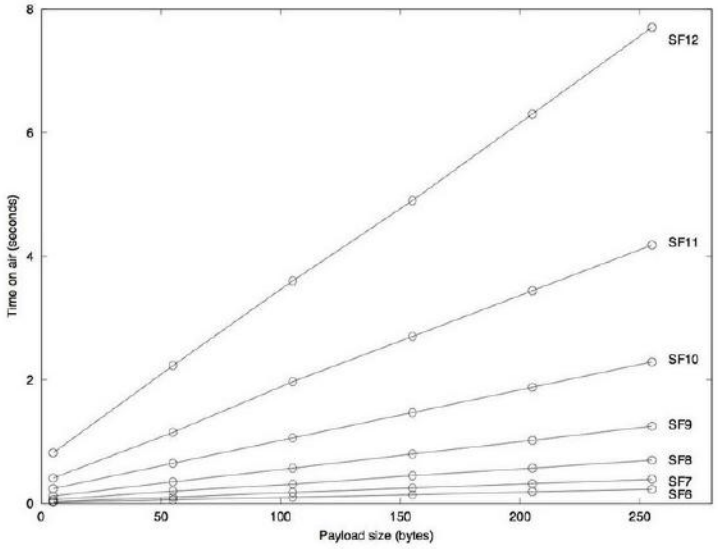


Fig 2.4 Time on Air for different spreading factors and payload size for 125 kHz band and 4/5 coding rate.

If we use different coding rate such as 4/8 we need to modify the obtained value with the 8/5 factor.

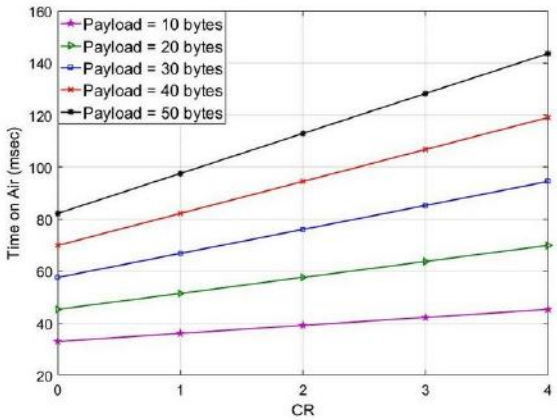


Fig 2.5 Time on Air for different payload size and coding rate (CR=1:5/4 to CR=4:8/4).

In the proposed example code we send 20-byte payload frame.

Note that our DevKit may be completed with a small solar providing max 100 mA (6V) cell such as:

As we need about 200 µA of continuous power supply only 2% of the cell efficiency is enough to keep our terminal node running.

2.3.1 Terminal and the gateway codes

The terminal part of this code contains a number of timing parameters. The initial values for time to sleep and time to wake up. Consequently, the time to wake up value is modified with the received timeout value provide by the gateway. The time till sleep is fixed and is related to the timing of the sensor capture and the time necessary to wait for the ACK packet. ACK packet contains new time to wake up value (if not equal to 0).

- `uint32_t timetillsleep=5000;`
- `uint32_t timetillwakeup=12000;`

The following sequence of instructions determines the reception phase during the **high-power period** (time to sleep).

```
      long debut,del=2000;
   // the receiver must respond in this waiting period - reception window
..
      turnOnRGB(COLOR_SEND,0);
      debut=millis();delay(300);turnOffRGB();Radio.Rx(0);
      while(millis()<(debut+del))
        {
        Radio.IrqProcess( ); delay(100);
        }
```

2.3.1.1 The code of terminal node

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#define timetillsleep 1000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

char txPacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );
```

```
typedef enum {LOWPOWER,ReadVoltage,TX} States_t;

States_t state;

bool sleepMode = false;
int16_t rssi,rxSize;
uint16_t voltage;

void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;       turnOffRGB(); //Radio.Sleep();
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup() {
    Serial.begin(9600);
    voltage = 0;
    rssi=0;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );

    state=ReadVoltage;
    Radio.Sleep( );  // LoRa modem sleep mode
    TimerInit( &sleep, onSleep );
    TimerInit( &wakeUp, onWakeUp );
    onSleep();
}

void loop()
{
if(lowpower)
  {
    lowPowerHandler();
  }
else
  {
  switch(state)
  {
    case TX:
    {
      sprintf(txPacket,"%s","ADC_battery (mV): ");
      sprintf(txPacket+strlen(txPacket),"%d",voltage);
      if(voltage<(uint16_t)3680)turnOnRGB(COLOR_SEND,0);
      else turnOnRGB(COLOR_RECEIVED,0);
      Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txPacket, strlen(txPacket));
      Radio.Send( (uint8_t *)txPacket, strlen(txPacket) );
      state=LOWPOWER;delay(100);
      break;
    }
    case LOWPOWER:
    {
      Serial.println("going to low power mode");
      delay(100);
      turnOffRGB();
      delay(100);  //LowPower time
      state = ReadVoltage;
      break;
    }
```

```
    case ReadVoltage:
    {
      Serial.println("reading battery voltage");
      pinMode(VBAT_ADC_CTL,OUTPUT);
      digitalWrite(VBAT_ADC_CTL,LOW);
      voltage=analogRead(ADC)*2;
      pinMode(VBAT_ADC_CTL, INPUT);
      state = TX;
      break;
    }
     default:
          break;
  }
  Radio.IrqProcess();
 }
}

void OnTxDone( void )
{
  Serial.println("TX done!");
  turnOffRGB();Radio.Sleep( );
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.println("TX Timeout......");
    state=ReadVoltage;
    Serial.println(state);
}
```

2.3.1.2 The code of receiver node

The corresponding node receives the data sent by the terminal and sends an **ACK packet** with new `timeout` value generated randomly. The node has no capacity to relay the received data to the IoT server via an Internet connection.
This kind of nodes are studied in the following Lab 3 related to multi-platform IoT architectures based on Low Power IoT DevKit and PYCOM-V IoT DevKit.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
/*
 * set LoraWan_RGB to 1,the RGB active in loraWan
 * RGB red means sending;
 * RGB green means received done;
 */
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );
void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr );
```

```
typedef enum
{
    LOWPOWER,
    RX,
    TX
} States_t;

int16_t txNumber;
States_t state;
bool sleepMode = false;
int16_t Rssi,rxSize;

void setup() {
    Serial.begin(9600);
    txNumber=0;
    Rssi=0;

    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    RadioEvents.RxDone = OnRxDone;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
    Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                                   LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                                   LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   0, true, 0, 0, LORA_IQ_INVERSION_ON, true );
    state=TX;
}

long tout;

void loop()
{
  switch(state)
  {
    case TX:
      delay(100);
      txNumber++;
      tout=10000 + random(10000);
        sprintf(txpacket,"%d",tout);
        turnOnRGB(COLOR_SEND,0);   // red
        Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txpacket, strlen(txpacket));
        Radio.Send( (uint8_t *)txpacket, strlen(txpacket) );
        state=LOWPOWER;
        break;
    case RX:
      Serial.println("into RX mode");
        Radio.Rx( 0 );
        state=LOWPOWER;
        break;
    case LOWPOWER:
      lowPowerHandler();
        break;
        default:
            break;
  }
    Radio.IrqProcess( );
}

void OnTxDone( void )
{
  Serial.print("TX done......");
  turnOnRGB(0,0);
  state=RX;
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.print("TX Timeout......");
    state=TX;
}
```

```
void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    Rssi=rssi;
    rxSize=size;
    memcpy(rxpacket, payload, size );
    rxpacket[size]='\0';
    turnOnRGB(COLOR_RECEIVED,0);  // green
    Radio.Sleep( );
    Serial.printf("\r\nreceived packet \"%s\" with Rssi %d , length %d\r\n",rxpacket,Rssi,rxSize);
    Serial.println("wait to send next packet");
    state=TX;
}
```

The following is a fragment of printout at the terminal side. Note that the **wake_up timer** is set with the received timeout value provided by the gateway.

```
..
packet send
TX done!

received packet: 10874 rssi: -39 , length: 5
Going into lowpower mode, 10874 ms later wake up.
Woke up, 5000 ms later into lowpower mode.
Humidity(%RH): 65.78     Temperature(C): 21.67
T:21.67, H:65.78

packet send
TX done!

received packet: 13872 rssi: -38 , length: 5
Going into lowpower mode, 13872 ms later wake up.
Woke up, 5000 ms later into lowpower mode.
Humidity(%RH): 65.76     Temperature(C): 21.67
T:21.67, H:65.76

packet send
TX done!

received packet: 11644 rssi: -36 , length: 5
Going into lowpower mode, 11644 ms later wake up.
Woke up, 5000 ms later into lowpower mode.
Humidity(%RH): 66.01     Temperature(C): 21.67
T:21.67, H:66.1

packet send
TX done!

received packet: 14145 rssi: -35 , length: 5
Going into lowpower mode, 14145 ms later wake up.

..
```

To do:

- Test the above codes
- Use micrometer to read the actual current required by the board in different operational periods and phases
- Modify the radio parameters such as spreading factor, signal bandwidth and the transmission power and observe the current consumption

Lab 3 – LoRa/WiFi gateways for CubeCell terminals

In this lab we are going to introduce the complete multi-platform IoT architectures including terminal nodes and the gateway nodes allowing us to communicate with the IoT servers.

We have two ways to build this kind of gateways:

1. By introducing the **PYCOM-V** based LoRa-WiFi gateway
2. By extending the capacity of the receiver node with an additional **WiFi modem**. This can be done by adding an ESP32 board to our kit and connect both MCU**s** via an **UART** link.

3.1 PYCOM-V LoRa sender and receiver

In this section we present the functionalities of the LoRa modem available for PYCOM-V DevKit.

3.1.1 LoRa modulation

As you probably know, LoRa modulation has three parameters:

- **`freq`** – frequency or carrier frequency from 868 to 870 MHz,
- **`sf`** – spreading factor or spreading of the spectrum or the number of modulations per bit sent (64-4096 expressed in powers of **2^7 to 2^{12}**)
- **`sb`** – signal bandwidth (31250 Hz to 500KHz)

By default we will use: **`freq=434MHz`** or 8**`68.5 MHz`**, **`sf=9`**, and **`sb=125KHz`**

Our DevKiT PYCOM-V can be completed with an expansion board – LoRa modem.

3.1.2 `sx127x.py` library

The **`sx127x.py`** library makes it possible to integrate the functionalities of the **sx1276/8 modem** into our applications.

This modem-circuit is connected to our base board by the **SPI** bus. An SPI bus operates on **3 basic lines** (signals): **SCK** – clock, **MISO** – Master_In_Slave_Out, **MOSI** – Master_Out_Slave_In, and on three control lines: **NSS** – Slave output selection or activation, RST – initialization signal, and **DIO0/INT** – interrupt signal sent by the activated Slave.

Here are some excerpts from the **`sx127x.py`** library

```
class SX127x:

    default_parameters = {
        "frequency": 869525000,
        "frequency_offset": 0,
        "tx_power_level": 14,
        "signal_bandwidth": 125e3,
        "spreading_factor": 9,
        "coding_rate": 5,
        "preamble_length": 8,
        "implicitHeader": False,
        "sync_word": 0x12,
        "enable_CRC": True,
        "invert_IQ": False,
    }
```

The default (radio) settings can be changed through the functions available in the same library:

```
        self.setFrequency(self.parameters["frequency"])
        self.setSignalBandwidth(self.parameters["signal_bandwidth"])

        # set LNA boost
        self.writeRegister(REG_LNA, self.readRegister(REG_LNA) | 0x03)
        # set auto AGC
        self.writeRegister(REG_MODEM_CONFIG_3, 0x04)
        self.setTxPower(self.parameters["tx_power_level"])
        self.implicitHeaderMode(self.parameters["implicitHeader"])
        self.setSpreadingFactor(self.parameters["spreading_factor"])
        self.setCodingRate(self.parameters["coding_rate"])
```

```
        self.setPreambleLength(self.parameters["preamble_length"])
        self.setSyncWord(self.parameters["sync_word"])
        self.enableCRC(self.parameters["enable_CRC"])
        self.invertIQ(self.parameters["invert_IQ"])
```

3.1.4 Main program

In the program that we are going to study we find all the parameters, the initialization and operating functions of the LoRa module.

In the **`lora_default`** list we offer the parameters compatible with our LoRa modems (ISM:434MHz/864MHz).

For now we will only focus on 3 parameters:

- **frequency**,
- **signal bandwidth** and
- the **spreading factor**

The modem is connected on the SPI bus; the **`lora_pins`** list identifies the signal numbers used to connect our modem to the PYCOM-V board.
Finally, the parameters and control signals associated with the **SPI** bus – **`lora_spi`** are determined.
With all the parameters initialized, communication between the card and the LoRa modem (SX127x) is activated.

Once the connection is activated we can call different **LoRa communication functions**:

```
        # type = 'sender'
        # type = 'receiver'
        # type = 'ping_master'
        # type = 'ping_slave'
        # type = 'receiver_callback'
```

Here is the complete code:

```
from machine import Pin,SPI
from sx127x import SX127x
from time import sleep
import LoRaSender
import LoRaReceiver
import LoRaPing
import LoRaReceiverCallback

# radio - modulation parameters
lora_default = {
    'frequency': 868500000,   #869525000,
    'frequency_offset':0,
    'tx_power_level': 14,
    'signal_bandwidth': 125e3,
    'spreading_factor': 9,
    'coding_rate': 5,
    'preamble_length': 8,
    'implicitHeader': False,
    'sync_word': 0x12,
    'enable_CRC': True,
    'invert_IQ': False,
    'debug': False,
}
# pins for PYCOM-V  on SPI bus
lora_pins = {
    'dio_0':3,
    'ss':7,       # 16 on SPI-LoRa ext. card
    'reset':8,    # RST
    'sck':4,
    'miso':6,
    'mosi':5,
}

lora_spi = SPI(
    baudrate=10000000, polarity=0, phase=0,
    bits=8, firstbit=SPI.MSB,
```

```
    sck=Pin(lora_pins['sck'], Pin.OUT, Pin.PULL_DOWN),
    mosi=Pin(lora_pins['mosi'], Pin.OUT, Pin.PULL_UP),
    miso=Pin(lora_pins['miso'], Pin.IN, Pin.PULL_UP),
)

lora = SX127x(lora_spi, pins=lora_pins, parameters=lora_default)

# type = 'sender'
type = 'receiver'
# type = 'ping_master'
# type = 'ping_slave'
# type = 'receiver_callback'

if __name__ == '__main__':
    if type == 'sender':
        LoRaSender.send(lora)
    if type == 'receiver':
        LoRaReceiver.receive(lora)
    if type == 'ping_master':
        LoRaPing.ping(lora, master=True)
    if type == 'ping_slave':
        LoRaPing.ping(lora, master=False)
    if type == 'receiver_callback':
        LoRaReceiverCallback.receiveCallback(lora)
```

3.1.5 LoRa functional modules

3.1.5.1 Sender – (`LoRaSender.py`)

Our transmitter module (sender) uses the OLED screen to present the value of the LoRa message counter. Data is sent as character strings.

```
from time import sleep
import machine, ssd1306
from machine import Pin, SoftI2C
import esp32

def disp(c):
    i2c = SoftI2C(scl=Pin(0), sda=Pin(10), freq=100000)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
    oled.fill(0)
    oled.text("SmartComputerLab", 0, 0)
    oled.text("LoRa sender", 0, 16)
    oled.text("Packet Nr:", 0, 32)
    oled.text(str(c), 0, 48)
    oled.show()


def send(lora):
    print("LoRa Sender")
    counter = 0
    while True:
        payload = 'Long long Hello ({0})'.format(counter)
        print('TX: {}'.format(payload))
        lora.println(payload)
        counter += 1
        disp(counter)
        sleep(5)

def send_payload(lora,payload):
    print('TX: {}'.format(payload))
    lora.println(payload)
```

To do:

1. Test the main program with the **`LoRaSender.py`** module above.
2. Add the reading of a sensor (**`sht21.py`**) and the sending of the sensed value.
3. Save the main program (**3.1.4**) as **`main.py`** , run it, then detach the board from your PC to run on battery power.

3.1.5.2 Receiver (LoRaReceiver.py)

Our receiver module (receive()) uses the OLED screen to present the **RSSI** value corresponding to the received LoRa messages.

```
from time import sleep
import machine, ssd1306
from machine import Pin, SoftI2C
import esp32

def disp(p):
    i2c = SoftI2C(scl=Pin(0), sda=Pin(10), freq=100000)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
    oled.fill(0)
    oled.text("SmartComputerLab", 0, 0)
    oled.text("LoRa receiver", 0, 16)
    oled.text("Packet Nr:", 0, 32)
    oled.text("RX:{}".format(p), 0, 48)
    oled.show()

def receive(lora):
    print("LoRa Receiver")

    while True:
        if lora.receivedPacket():
            try:
                payload = lora.readPayload().decode()
                rssi = lora.packetRssi()
                print("RX: {} | RSSI: {}".format(payload, rssi))
                disp(rssi)
            except Exception as e:
                print(e)
```

To do:

1. Test the main program with the **`LoRaReceiver.py`** module above.
2. Add payload value presentation on OLED screen.
3. Save the main program(**3.1.4**) as **`main.py`** , run it, then detach the board from your PC to run on battery power.

3.1.5.3 Receiver – onReceive (LoRaReceiverCallback.py)

The reception of a LoRa packet can be performed **asynchronously** by means of the **interrupt signal** generated by the **sx127x** modem (**INT/DIO0**) at the time of reception of the physical frame and its recording in the reception buffer.

Here is the code:

```
from time import sleep
import machine, ssd1306
from machine import Pin, SoftI2C
import esp32

def disp(p):
    i2c = SoftI2C(scl=Pin(0), sda=Pin(10), freq=100000)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
    oled.fill(0)
    oled.text("SmartComputerLab", 0, 0)
    oled.text("LoRa onReceive", 0, 16)
    oled.text("Packet Nr:", 0, 32)
    oled.text("RX:{}".format(p), 0, 48)
    oled.show()

def receiveCallback(lora):
    print("LoRa Receiver Callback")
```

```
    lora.onReceive(onReceive)
    lora.receive()


def onReceive(lora, payload):
    try:
        payload = payload.decode()
        rssi = lora.packetRssi()
        print("RX: {} | RSSI: {}".format(payload, rssi))
        disp(rssi)
    except Exception as e:
        print(e)
```

To do:

As in the previous example.

3.2 PYCOM-V LoRa-WiFi gateways

In this section we will develop an architecture integrating several essential devices for the creation of a complete IoT system. The central device will be the gateway between the LoRa links and the WiFI communication.

3.2.1 PYCOM-V LoRa-WiFi (MQTT) gateway

Our first example illustrates the construction of a LoRa-WiFi gateway to an **MQTT** broker.
The gateway (G – gateway) receives the LoRa packets with a payload containing the data from the sensors attached to LoRa terminal (T).

The modules to import include:

```
from umqtt.robust import MQTTClient
```

This module is used to define the broker (its IP address) and to establish a TCP connection on port 1883. (unsecured version)

WiFi connection is realized by our **wifista** module; this module can be modified in order to be able to choose between a static or dynamic address.

We use two serial buses: **SPI** and **I2C** (soft version). The OLED screen is attached to the SoftI2C bus.

The LoRa modem is connected by the SPI bus whose parameters are defined in the code. The default radio settings are also set in code (l**ora_default**).

```
from umqtt.robust import MQTTClient
import machine
from machine import Pin,SPI,SoftI2C
import ssd1306
import wifista
import utime as time
import gc
from sx127x import SX127x

def disp(p):
    i2c = SoftI2C(scl=Pin(0), sda=Pin(10), freq=100000)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
    oled.fill(0)
    oled.text("SmartComputerLab", 0, 0)
    oled.text("LoRa receiver", 0, 16)
    oled.text("Packet Nr:", 0, 32)
    oled.text("RSSI: {}".format(p), 0, 48)
    oled.show()


# radio - modulation parameters
lora_default = {
    'frequency': 868500000,
    'frequency_offset':0,
    'tx_power_level': 14,
    'signal_bandwidth': 125e3,
    'spreading_factor': 9,
    'coding_rate': 5,
    'preamble_length': 8,
    'implicitHeader': False,
    'sync_word': 0x12,
    'enable_CRC': True,
    'invert_IQ': False,
    'debug': False,
}

# modem - connection wires-pins on SPI bus
# pins for PYCOM-V

lora_pins = {
    'dio_0':3,
    'ss':7,       # 16 on SPI-LoRa ext. card
    'reset':8,    #RST
    'sck':4,
```

```
    'miso':6,
    'mosi':5,
}

lora_spi = SPI(
    baudrate=10000000, polarity=0, phase=0,
    bits=8, firstbit=SPI.MSB,
    sck=Pin(lora_pins['sck'], Pin.OUT, Pin.PULL_DOWN),
    mosi=Pin(lora_pins['mosi'], Pin.OUT, Pin.PULL_UP),
    miso=Pin(lora_pins['miso'], Pin.IN, Pin.PULL_UP),
)

lora = SX127x(lora_spi, pins=lora_pins, parameters=lora_default)

wifista.connect()
broker = "broker.emqx.io"
client = MQTTClient("PYCOM-V", broker)

def only_publish():
    count = 1
    rssi =0
    while True:
        if lora.receivedPacket():
            payload = lora.readPayload().decode()
            rssi = lora.packetRssi()
            print("RX: {} | RSSI: {}".format(payload, rssi))
            mess="RSSI: " + str(rssi)
            wifista.connect()
            client.publish(b"pycom-v/test", mess)
            count = count + 1
            disp(rssi)
            time.sleep(10)

client.reconnect()
only_publish()
```

To do:

1. Test the above program.
2. Retrieve the **payload value** (data received from the **SHT21** sensor and the **state of battery in millivolts**) and send it in the MQTT message.

3.2.2 PYCOM-V LoRa-WiFi (ThingSpeak) gateway

The LoRa-WiFi gateway (ThingSpeak) will retransmit the data received on a LoRa link and send it over a WiFi connection to a ThingSpeak server.

The following program allows you to receive LoRa packets and retransmit it over a WiFi line to the ThingSpeak server. We use here the notion of **topic** (MQTT) and messages.

The topic is a **character string** including the **channel number**, the **publish command** and the **write key**.

```
topic = "channels/" + CHANNEL_ID + "/publish/" + WRITE_API_KEY
```

The message is the payload with the **fields** associated with each value:

```
payload = "field1="+str(temp)+"&field2="+str(hum)+"&field3="+str(rssi)
```

Here is the full code:

```
from umqtt.robust import MQTTClient
import machine
from machine import Pin,SPI,SoftI2C
import ssd1306
import wifista
import utime as time
import gc
from sx127x import SX127x

def disp(p):
    i2c = SoftI2C(scl=Pin(0), sda=Pin(10), freq=100000)
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)
    oled.fill(0)
    oled.text("SmartComputerLab", 0, 0)
    oled.text("LoRa receiver", 0, 16)
    oled.text("Packet Nr:", 0, 32)
    oled.text("RSSI: {}".format(p), 0, 48)
    oled.show()

# radio - modulation parameters
lora_default = {
    'frequency': 434500000,   #869525000,
    'frequency_offset':0,
    'tx_power_level': 14,
    'signal_bandwidth': 125e3,
    'spreading_factor': 9,
    'coding_rate': 5,
    'preamble_length': 8,
    'implicitHeader': False,
    'sync_word': 0x12,
    'enable_CRC': True,
    'invert_IQ': False,
    'debug': False,
}

# pins for PYCOM-V on SPI bus

lora_pins = {
    'dio_0':3,
    'ss':7,      # 16 on SPI-LoRa ext. card
    'reset':8,  #RST
    'sck':4,
    'miso':6,
    'mosi':5,
}

lora_spi = SPI(
    baudrate=10000000, polarity=0, phase=0,
    bits=8, firstbit=SPI.MSB,
    sck=Pin(lora_pins['sck'], Pin.OUT, Pin.PULL_DOWN),
    mosi=Pin(lora_pins['mosi'], Pin.OUT, Pin.PULL_UP),
    miso=Pin(lora_pins['miso'], Pin.IN, Pin.PULL_UP),
)
```

```
lora = SX127x(lora_spi, pins=lora_pins, parameters=lora_default)

wifista.connect()
server = "mqtt.thingspeak.com"
client = MQTTClient("umqtt_client", server)
CHANNEL_ID = "1626377"
WRITE_API_KEY = "3IN09682SQX3PT4Z"

def only_publish():
    topic = "channels/" + CHANNEL_ID + "/publish/" + WRITE_API_KEY
    temp =21.5
    hum =55.7
    count = 1
    rssi =0
    while True:
        if lora.receivedPacket():
            buff = lora.readPayload().decode()
            rssi = lora.packetRssi()
            print("RX: {} | RSSI: {}".format(buff, rssi))
            mess="RSSI: " + str(rssi)
            wifista.connect()
            payload = "field1="+str(temp)+"&field2="+str(hum)+"&field3="+str(rssi)
            client.connect()
            client.publish(topic, payload)
            client.disconnect()
            temp=temp+1.0
            hum=hum+2.0
            time.sleep(15)


only_publish()
```

To do:

1. Test the above program.
2. Retrieve the **payload value** (data received from the **SHT21** sensor and the **state of battery in millivolts**) and send it in the MQTT message.
3. Write the same application with the reception of LoRa packets by the callback function (interrupt)

3.3 Simple UART connection to WiFi gateway with ESP32

Let us show how to connect a CubeCell board with an ESP32 board using software serial. The following code example runs on CubCell board. It generates four numbers and outputs them on a **softSerial** link.

3.3.1 Sending data from CubeCell via an UART link

```
#include "softSerial.h"
softSerial ss(GPIO1 /*TX pin*/, GPIO2 /*RX pin*/);  //
//softSerial ss(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);  // GPIO5 (33) , GPIO3 (8)

void setup()
{
  Serial.begin(9600);delay(2000);
  Serial.println();
  Serial.println("Hardware Serial init");
  ss.begin(9600);delay(2000);
}

float s1=0.1,s2=0.2,s3=0.3,s4=0.4;
union
{
uint8_t frame[16];
float sensor[4];
} sdp; // send data packet

void loop()
{
  Serial.println("send...");
  s1+=1.0;s2+=1.0;s3+=1.0;s4+=1.0;
  sdp.sensor[0]=s1;sdp.sensor[1]=s2;sdp.sensor[2]=s3;sdp.sensor[3]=s4;
  for(int i=0;i<16;i++) ss.sendByte(sdp.frame[i]);
  delay(2000);
}
```

3.3.2 Receiving data from CubeCell via an UART link and sending them to ThingSpeak

On the ESP32 board side we use **HardwareSerial**. The board receives the data on the **HardwareSerial** link and sends them to the **ThingSpeak** server. The ESP32 board is connected to **ThingSpeak** server via a WiFi link. The received (via **uart**) data are sent to the ThingSpeak server every 20 seconds.

```
#include <Arduino.h>
#include <ThingSpeak.h>
#include <WiFi.h>
HardwareSerial uart(2); // 16 -RX,  17-TX

char* ssid     = "PhoneAP";
char* pass  = "smartcomputerlab";

WiFiClient client;
unsigned long myChannelNumber = 114;   // Thinspeak channel
const char *myWriteAPIKey="E8KYBCRD2Z59LVWJ" ;  // write code

union
{
uint8_t frame[16];
float sensor[4];
} rdp; // send data packet

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  delay(2000);
  Serial.println();
  Serial.println("Goodnight moon!");
  // set the data rate for the SoftwareSerial port
  uart.begin(9600);  delay(2000);
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
}
```

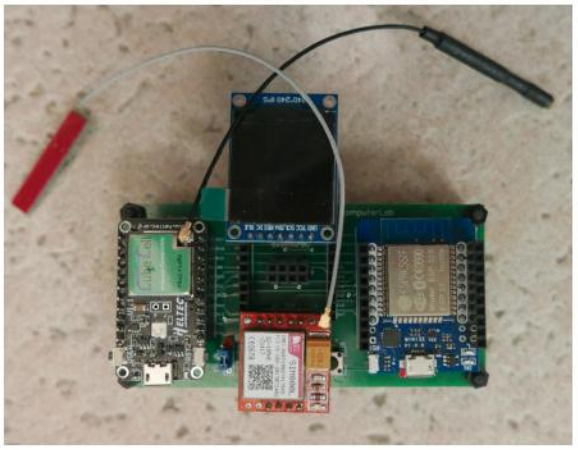
```
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");Serial.println(ip);
ThingSpeak.begin(client);
}

char recv;
long lasttime=0;
int i=0;

void loop()
{
if (uart.available())
  {
  recv=uart.read(); rdp.frame[i]=(uint8_t) recv; i++;
  if(i>15)
    {
    Serial.println(rdp.sensor[0]);Serial.println(rdp.sensor[1]);
    Serial.println(rdp.sensor[2]);Serial.println(rdp.sensor[3]);
    Serial.println();i=0;
    if(millis()-lasttime>20000)  //the data may be sent every 20 seconds
      {
      Serial.println("ThingSpeak begin");
      Serial.println("Fields update");
      ThingSpeak.setField(1, rdp.sensor[0]);
      ThingSpeak.setField(2, rdp.sensor[1]);
      ThingSpeak.setField(3, rdp.sensor[2]);
      ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
      lasttime=millis();
      }
    }
  }
  else
    {
    delay(1000);
    i=0;
  }
}
```

3.3.3 LoRa receiver on CubeCell and WiFi gateway with ESP32

In the following example we show a complete IoT architecture with one terminal node (CellCube) and one gateway board with CellCube receiver and a UART bridge to ESP32 .



3.3.3.1 CubeCell low power LoRa sender (to gateway)

```
#include "Arduino.h"
#include "LoRaWan_APP.h"
#include <Wire.h>
#include "SHT21.h"
SHT21 SHT21;
//#define timetillsleep 5000
//#define timetillwakeup 12000
```

```
uint32_t timetillsleep=5000;
uint32_t timetillwakeup=12000;
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1, highpower=0;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here


char txPacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];

static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );
int16_t txNumber;
int16_t rssi,rxSize;

union
{
uint8_t frame[16];
float sensor[4];
} sdp; // send data packet


void getSHT21()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);
    delay(50);
    Wire.begin(29,28);
    SHT21.begin();
    sdp.sensor[0]=SHT21.getTemperature();
    sdp.sensor[1]=SHT21.getHumidity();
    Wire.end();
    digitalWrite(Vext, HIGH);
}

void onSleep()
{
  Serial.printf("Going into lowpower mode, %d ms later wake up.\r\n",timetillwakeup);
  lowpower=1;highpower=0;
  //timetillwakeup ms later wake up;
  TimerSetValue( &wakeUp, timetillwakeup );
  TimerStart( &wakeUp );
}
void onWakeUp()
{
  Serial.printf("Woke up, %d ms later into lowpower mode.\r\n",timetillsleep);
  lowpower=0;highpower=1;
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleep, timetillsleep );
  TimerStart( &sleep );
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
```

```
  RadioEvents.RxDone = OnRxDone;
  RadioEvents.TxDone = OnTxDone;
  RadioEvents.TxTimeout = OnTxTimeout;
  Radio.Init( &RadioEvents );
  Radio.SetChannel( RF_FREQUENCY );
  Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                  LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                  LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                  true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
  Radio.Sleep( );
  TimerInit( &sleep, onSleep );
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}

long debut,del=2000;  // the receiver must respond in this waiting period
uint16_t voltage;

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  if(highpower)
     {
      char buffp[32];
      turnOnRGB(COLOR_SEND,0);
      getSHT21(); //strcpy(buff,"hello");
      voltage=getBatteryVoltage();
      sdp.sensor[2]=(float)((int)voltage);
      Serial.println(voltage);Serial.println(sdp.sensor[2]);
      Radio.Send( sdp.frame, 16);
      debut=millis();delay(300);turnOffRGB();Radio.Rx(0);
      while(millis()<(debut+del))
        {
        Radio.IrqProcess( ); delay(100);
        }
       highpower=0;
    }
}

uint32_t tout;

void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    rssi=rssi;
    rxSize=size;
    memcpy(rxpacket, payload, size );
    rxpacket[size]='\0';
    tout=atoi(rxpacket);
    turnOnRGB(COLOR_RECEIVED,0);
    timetillwakeup=tout;
    Radio.Sleep( );
    Serial.printf("\r\nreceived packet: %d rssi: %d , length: %d\n",tout,rssi,rxSize);delay(100);
    turnOffRGB();
}

void OnTxDone( void )
{
  Serial.println("TX done!");
  turnOffRGB();
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.println("TX Timeout......");
}
```

3.3.3.2 CubeCell LoRa receiver and UART bridge

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include "softSerial.h"
softSerial ss(GPIO1 /*TX pin*/, GPIO2 /*RX pin*/);
 /* set LoraWan_RGB to 1,the RGB active in loraWan
 * RGB red means sending;
 * RGB green means received done; */

#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868500000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       9         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];
static RadioEvents_t RadioEvents;
void OnTxDone( void );
void OnTxTimeout( void );
void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr );

typedef enum
{
    LOWPOWER,
    RX,
    TX
} States_t;

int16_t txNumber;
States_t state;
bool sleepMode = false;
int16_t Rssi,rxSize;

union
{
uint8_t frame[16];
float sensor[4];
} rdp; // send data packet

void setup() {
    Serial.begin(9600);
    ss.begin(9600);
    txNumber=0;
    Rssi=0;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    RadioEvents.RxDone = OnRxDone;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
    Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                                   LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                                   LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   0, true, 0, 0, LORA_IQ_INVERSION_ON, true );
    state=TX;
}
```

```
long tout;

void loop()
{
  switch(state)
  {
    case TX:
      delay(100);
      txNumber++;
      tout=10000 + random(10000);   // send random timeout
      sprintf(txpacket,"%d",tout);
      turnOnRGB(COLOR_SEND,0);  // red
      Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txpacket, strlen(txpacket));
      Radio.Send( (uint8_t *)txpacket, strlen(txpacket) );
      state=LOWPOWER;
      break;
    case RX:
      Serial.println("into RX mode");
      Radio.Rx(0);
      state=LOWPOWER;
      break;
    case LOWPOWER:
      lowPowerHandler();
      break;
    default:
      break;
  }
    Radio.IrqProcess( );
}

void OnTxDone( void )
{
  Serial.print("TX done......");
  turnOnRGB(0,0);
  state=RX;
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.print("TX Timeout......");
    state=TX;
}


void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    Rssi=rssi;
    rxSize=size;
    memcpy(rdp.frame, payload, size );
    //rxpacket[size]='\0';
    turnOnRGB(COLOR_RECEIVED,0);  // green
    for(int i=0;i<size;i++) ss.sendByte(rdp.frame[i]);
    Radio.Sleep( );
    Serial.printf("\r\nreceived packet with Rssi %d , length %d\r\n",Rssi,rxSize);
    Serial.println("wait to send next packet");
    state=TX;
}
```

The UART data are received by the ESP32 and sent to the **ThingSpeak** server (see **3.1.2)**

3.4 Assignment – terminal's scheduler in gateway node

In this assignment you have to implement a scheduler that will organize the emission (high_power) periods in the network of the co-operating terminals. Each terminal identified by its Chip_ID and accepted by the gateway has a simple number (order of arrival). To this number is allocated time slot for the next transmission.

The time-slot is a part of the gateway cycle (e.g. 600 seconds). For example if we have 10 active terminals each terminal obtains a slot of 60 seconds. The beginning of the slot (time) is sent to the terminal each times it sends the new data packet. The arrival of a new terminal and its random transmission (if received and accepted) modifies the scheduler table adding a new slot. Next cycle all terminals (e.g.11 terminals) are dispatched in equal slots of 600/11 seconds each.

Implement the above algorithm and test the whole IoT architecture with more than 2 terminals.

The terminal node may receive an additional parameter called delta. The delta parameter provides the degree of precision for the sensor values expressed in %. The terminal compares the previous sensor value(s) with the actual one and in the case when the difference between the previous_value and the new_value is smaller than previous_value * delta , there is no need to send the new LoRa package.
This may done once or twice with the calculation of the new timeout value by adding one or two operational cycles.