Low Power IoT Labs:

Building complete IoT Architectures with Terminals, Gateways, and Servers

In the following labs – Lab1 and Lab2 we are going to study and experiment with complete IoT architectures including terminal nodes (T) long range gateways (G) and IoT servers. To support the experimentation we are using multiple IoT platforms from **SmartComputerLab**.

Table of Contents

In the following labs – Lab3 and Lab4 we are going to study and experiment with complete IoT architectures including terminal nodes (T) long range gateways (G) and IoT servers. To support the experimentation we are using multiple IoT platforms from **SmartComputerLab**.
These are

- CubeCell (ASR6501) platform with **ARM-Cortex M0** and **SX1262** LoRa modem
- ESP32 (Heltec) platform with WiFi – BT/BLE and **SX1276** LoRa modem
- Raspberry-Pi 3 and 4 boards with RAK2245 – **SX1301** multichannel LoRa modem

Both the CubeCell and ESP32 (Heltec) platforms operate with Nordic PPK2 – Power Profiler Kit II

During the first Lab3 we are introducing the CubeCell platform and first we are going to experiment with the board itself and different sensors (and display) attached to I2C bus. Then we will experiment with different communication functions related to LoRa modem including different radio parameters.
In this context we are going to use two boards in sender/receiver or client/server modes. The sender/client nodes operate in two phases :

- **high_power** (consumption) phase when sensors are activated and the radio transmission takes place.
- **low_power** phase when the sensors are de-activated (no power) and the MCU and LoRa modem enter into **deep-sleep state**

We will measure the value of the current flowing during these phases. To complete the measurements we are going to take into account different radio parameters for LoRa transmission such as signal bandwidth and spreading factor.

In the second part of this lab we will exploit the ESP32 (Heltec WiFi-LoRa V2) boards to build the LoRa-WiFi gateways and to communicate with the external IoT servers. In this context we will experiment with LoRaMQTT and LoRaTS protocols that are adapted directly to these applications.

These IoT gateways may be configured using different means, such as WiFi SmartConfig protocol, simple external EEPROM modules or micro-keyboards attached via UART interface.

During the second Lab4 we are going to study and experiment with **LoRaWAN** protocol and related gateways and devices. LoRaWAN is a standard protocol based on LoRa radio links and implemented by many telecom operators as well as by the open source communities.
We provide you with several LoRaWAN gateways combining Raspberry-Pi boards and SX1301 multichannel modems. In our case the RPI/SX1301 gateways will operate exclusively as packet forwarders.

We are going to exploit the LoRaWAN libraries running on CubeCell and ESP32 platforms to build some complete applications with several devices attached to each one.

You will be asked to analyze the power consumption of the terminal nodes (devices) and send the status of the battery to the The Things Network (TTN) network server. The TTN server is a kind of relay between the LoRaWAN gateways and the application servers such as ThingSpeak.

Lab 1 (Low Power IoT Labs)

Simple IoT Architectures with CubeCell and ESP32-WiFi-LoRa boards

1.1 Introduction

This Lab (3-9 hours) is prepared to teach the low power technologies for IoT architectures and to experiment with CubeCell boards and LoRaWAN IoT DevKit from SmartComputerLab.
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).
Our IoT DevKit uses ARM-Cortex M0 and LoRa SX1262 modem combined into ASR6501 SoM (System on Module). The ASR6501 module is integrated into the CubeCell board.

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 off** 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 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.

The following diagram shows the operational line of a terminal.

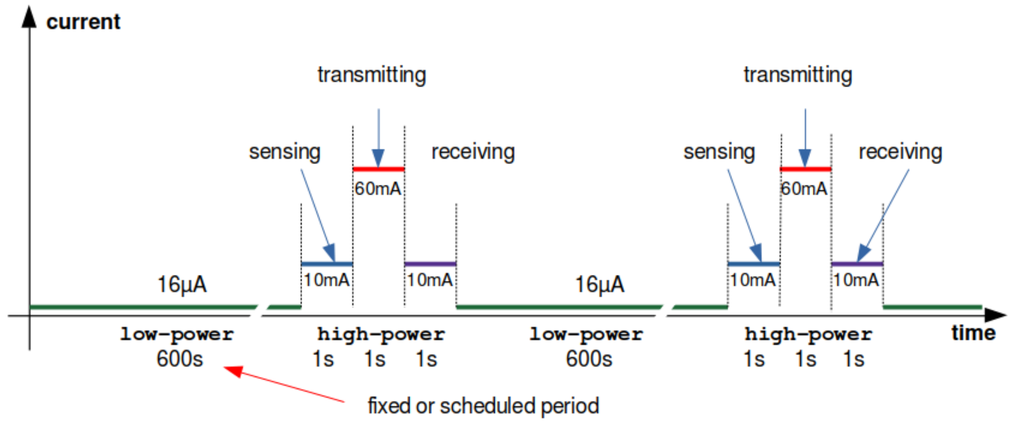


Figure 1.1 Low_power and **high_power** periods of operational cycle

1.2 Hardware Architecture - main board (CubeCell)

Heltec CubeCell board (HTCC-AB01) uses the ASR6501 module. The ASR6501 is SiP's (system-in-package) that combines a Cypress PSoC 4000 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.

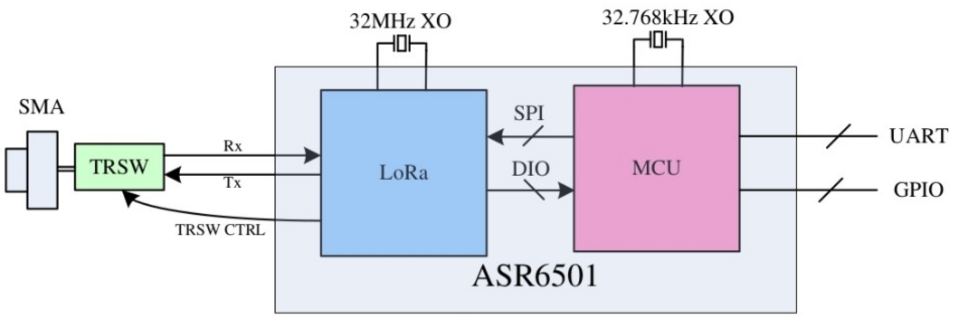


Figure 1.2 ASR6501 architectural scheme with MCU and Lora modem connected via SPI bus

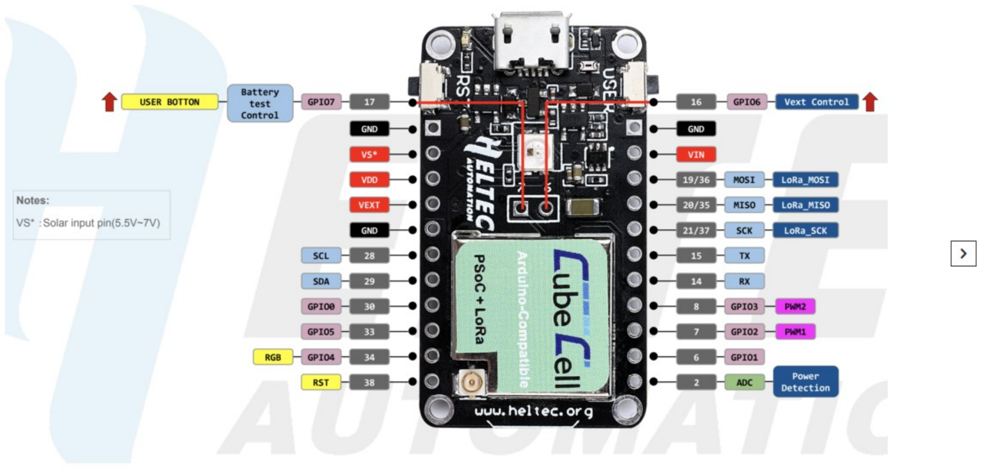


Figure 1.3 CellCube board and its pinout

for 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.

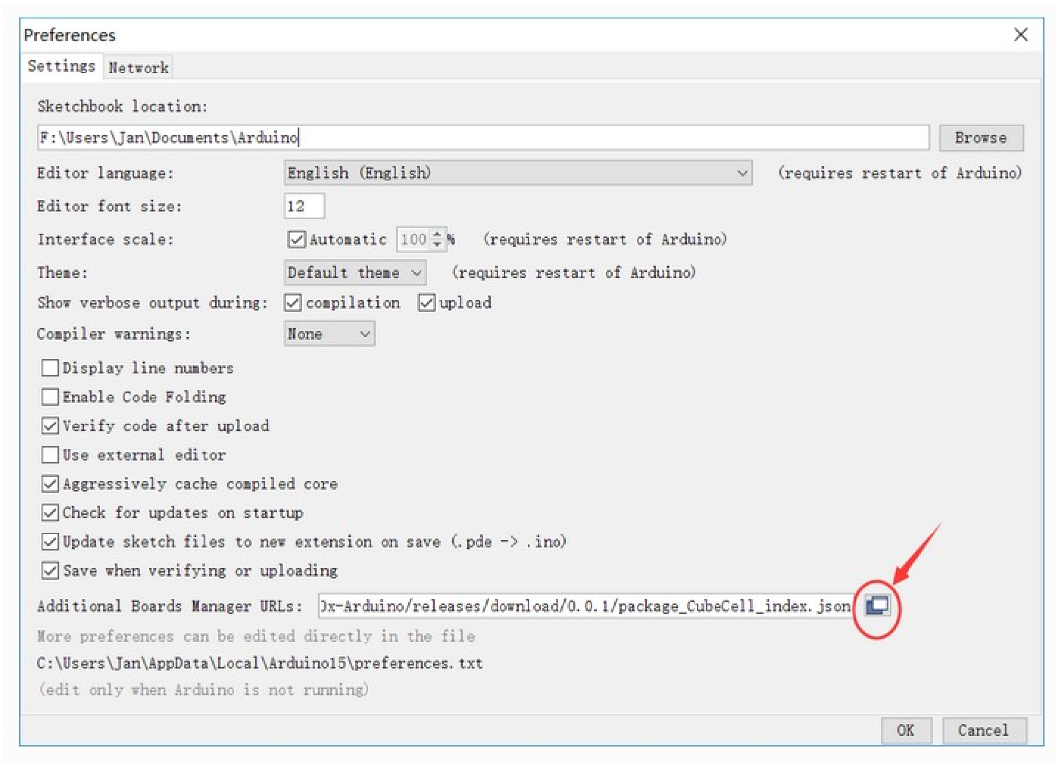
Even if we divide this result by 2 we still have 3 months of power supply.

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** (from **Ukraine**)was added. Heltec uses a custom CubeCell bootloader for ASR650x. Serial number and a license that enables Arduino support are stored in flash memory.

1.2.1 Software installation with the use Arduino board manager

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



Preferences

Settings | Network

Sketchbook location:
F:\Users\Jan\Documents\Arduino

Editor language: English (English) (requires restart of Arduino)
Editor font size: 12
Interface scale: ✓ Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: ✓ compilation ✓ upload
Compiler warnings: None

☐ Display line numbers
☐ Enable Code Folding
✓ Verify code after upload
☐ Use external editor
✓ Aggressively cache compiled core
✓ Check for updates on startup
✓ Update sketch files to new extension on save (.pde -> .ino)
✓ Save when verifying or uploading

Additional Boards Manager URLs:)x-Arduino/releases/download/0.0.1/package_CubeCell_index.json
More preferences can be edited directly in the file
C:\Users\Jan\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

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

1.2.2 Software installation via Git

- **`For Windows: https://github.com/HelTecAutomation/ASR650x-Arduino/blob/master/InstallGuide/windows.md`**
- **`For MacOS: https://github.com/HelTecAutomation/ASR650x-Arduino/blob/master/InstallGuide/mac.md`**
- **`For Linux(Ubuntu/Debian): https://github.com/HelTecAutomation/ASR650x-Arduino/blob/master/InstallGuide/debian_ubuntu.md`**

Principal link for software resources:

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

1.3 Power consumption test with Power Profiler Kit II

The **Power Profiler Kit II** (PPK2) from **NORDIC** Semiconductor is a **standalone unit**, which can measure and optionally supply currents all the way from sub-uA and as high as 1A on all SmartComputerLab IoT DevKit boards from Smartcomputerlab.

The PPK2 is powered via a standard 5V USB cable, which can supply up to 500mA of current. In order to supply up to 1A of current, two USB cables are required.

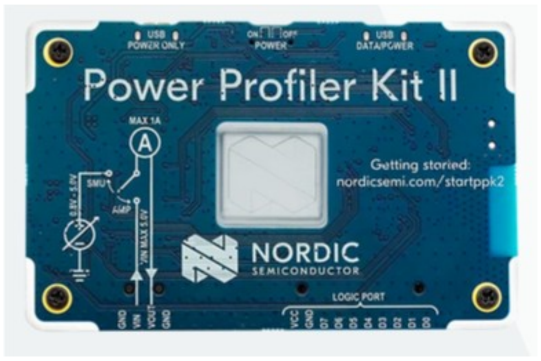
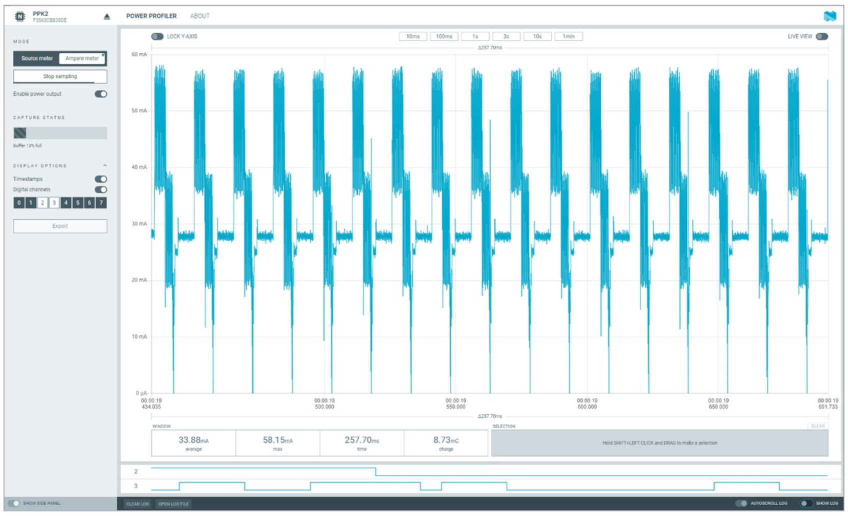


Fig 1.4 Power Profiler Kit II

An **ampere meter only mode (to be used in our case)** , as well as a **source mode** (shown as AMP and source measure unit (SMU) respectively on the PCB) are supported. For the ampere meter mode, an external power supply must source VCC levels between 0.8 and 5V to the device under test (DUT). For the source mode, the PPK2 supplies VCC levels between 0.8 and 5V and the on-board regulator supplies up to 1A of current to external applications.
It is possible to measure **low sleep** currents, the **higher active currents**, as well as **short current peaks.**

The PPK2 has an advanced analog measurement unit with a high dynamic measurement range. This allows accurate power consumption measurements for the entire range typically seen in low-power embedded applications, all the way from single µAs to 1A. The resolution varies between 100nA and 1mA depending on the measurement range and is high enough to detect small spikes as often seen in low power optimized systems.

The PPK2 can also use digital inputs as a low-end logic analyzer, enabling code-synchronized measurements. This can be achieved by connecting the digital inputs to an I/O pin on the external device under test (DUT).
In order to use this functionality, the DUT must be powered by a VCC voltage of 1.6-5.5V. The digital input can then show what code is executed in the DUT at different points in time.
10x quicker sampling (i.e. 100ksps) compared with the long term window of the previous generation enables maximum continuous resolution at all times. This enables a user to gather average acquisition data and zoom in for high resolution data using the same window.



33.88mA	58.15mA	257.70ms	8.73mC
average	max	time	charge

Fig 1.5 Screenshot of the **Power Profiler app** running in **ampere meter mode**.

The following figure shows the connection of the IoT DevKit with CubeCell board to the Power Profiler Kit II (PPK2). Note the use of JMP connector to provide **Vin**/**Vout** signal to the PPK2.

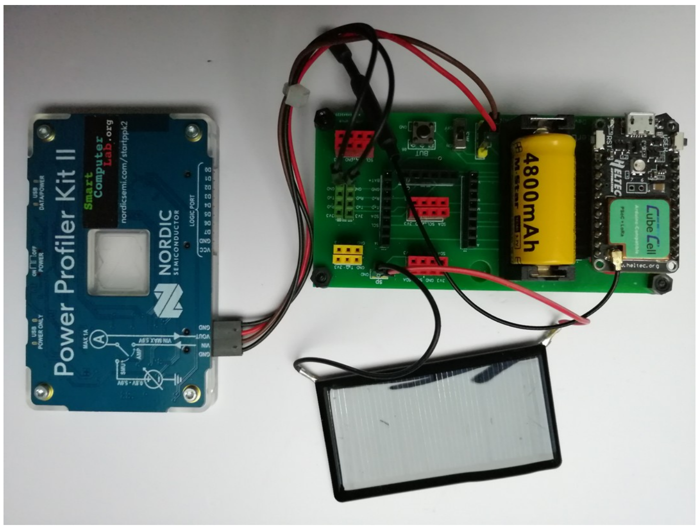


Fig 1.6 CubeCell IoT DevKit (on battery) connected to **PPK2** (**jumper** connector)and **solar panel**.

1.3.1 PPK2 software installation

The PPK2 software may be installed on Windows, MacOS and Linux OS. In our case we are using Ubuntu 20.04LTS.
In order to install PPK2 software on Ubuntu we need 3 software components.

1. The **application** in executable mode:

```
nrfconnect-3.11.0-x86_64.appimage
```

To be found in:

`https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-desktop/Download#infotabs`

2. The **`Jlink`** driver to be installed by:

```
sudo dpkg -i  Jlink_Linux_V758b_x86_64.deb
```

To be found in: (look for **V758b** version):
`https://www.segger.com/downloads/jlink/`

3. The **`nrf-udev`** package to be installed by:

```
sudo dpkg -i  nrf-udev_1.0.1-all.deb
```

To be found in:

https://github.com/NordicSemiconductor/nrf-udev/releases

1.4 Testing CubeCell HTCC-AB01 main board

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.

1.4.1 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);
}
```

1.4.2 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;
}
```

1.4.3 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() {}
```

1.4.4 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
```

1.4.5 Sleep modes: timer and user interruption

1.4.5.1 Timer mode

```
#include "Arduino.h"
#include "LoRaWan_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() {
  Serial.begin(9600);
  Radio.Sleep( );  // LoRa modem sleep mode
  TimerInit( &sleep, onSleep );
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}

int i=0;

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  Serial.print('.');i++;
  if(i==40) { i=0; Serial.println();}
  delay(400);
}
```

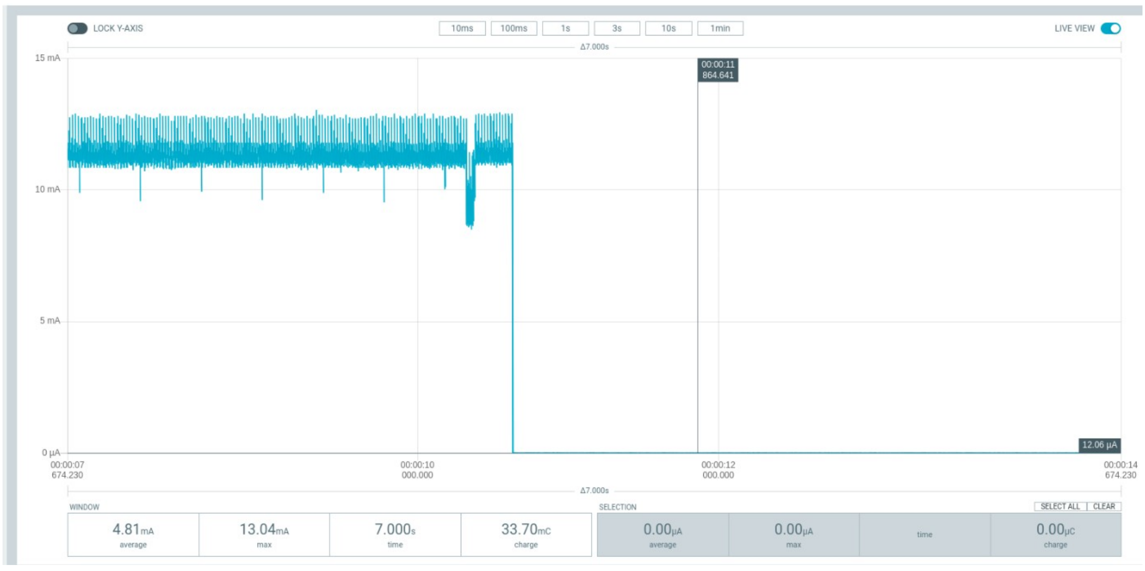
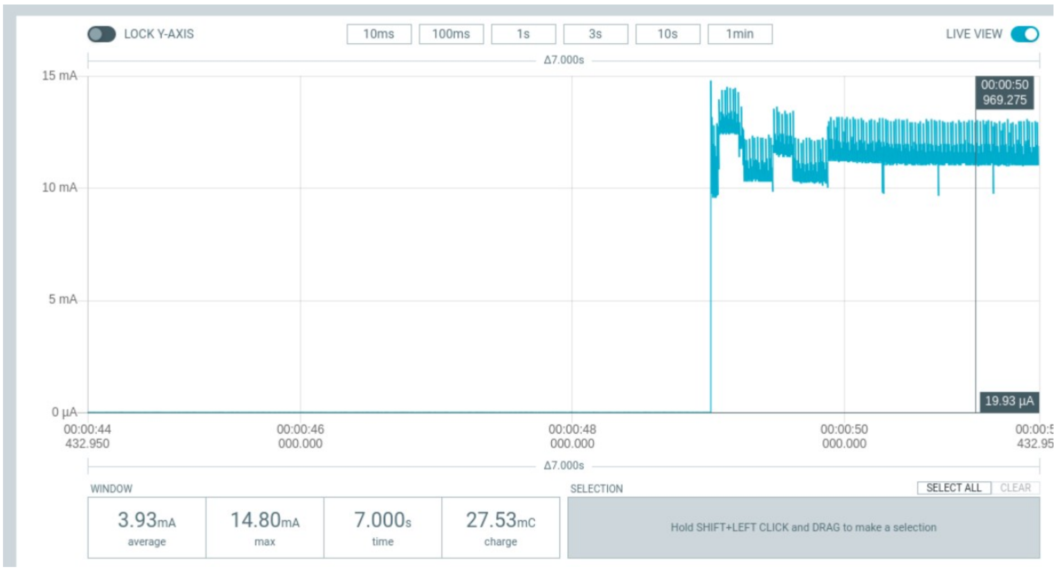


Fig 1.7 PPK2 display for deep-sleep mode: **max 13.04 mA** (**high-power**), min **12.06 µA** (**low-power**)

1.4.5.2 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() {
  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);
}

int i=0;

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  Serial.print('.');i++;
  if(i==40) { i=0; Serial.println();}
  delay(400);
}
```



3.93mA average	14.80mA max	7.000s time	27.53mC charge

Hold SHIFT+LEFT CLICK and DRAG to make a selection

Fig 1.8 PPK2 display for deep-sleep mode with user (USER button) interruption

1.4.6 System time

System time is base on internal timer counter that operates at milliseconds level (seconds/subseconds).

```
typedef struct TimerSysTime_s
 {
  uint32_t Seconds;
  int16_t SubSeconds;
   }
TimerSysTime_t;
```

The timer is operating also during the deep_sleep mode.

1.4.6.1 System time in active - high power mode

```
TimerSysTime_t sysTimeCurrent;
void setup()
{
  Serial.begin(9600);

  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() {

  delay(1000);
  sysTimeCurrent = TimerGetSysTime( );
  Serial.printf("sys time:%u.%d\r\n",(unsigned int)sysTimeCurrent.Seconds,
sysTimeCurrent.SubSeconds);
}
```

1.4.6.2 System time in deep_sleep - low power 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 );
  }
}

TimerSysTime_t sysTimeCurrent;

void setup() {
  Serial.begin(9600);
  pinMode(INT_GPIO,INPUT_PULLUP);
  attachInterrupt(INT_GPIO,onWakeUp,FALLING);
  TimerInit( &sleep, onSleep );
  Serial.printf("Going into lowpower mode. Press user key to wake up\r\n");
  delay(5);
}

int i=0;

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  sysTimeCurrent = TimerGetSysTime( );
  Serial.printf("sys time:%u.%d\r\n",(unsigned int)sysTimeCurrent.Seconds,
sysTimeCurrent.SubSeconds);
  Serial.print('.');i++;
  if(i==40) { i=0; Serial.println();}
  delay(1000);
}
```

Execution example:

```
.Going into lowpower mode. Press user key to wake up
Woke up by GPIO, 5000 ms later into lowpower mode.
sys time:25.951
.sys time:26.991
.sys time:28.12
.sys time:29.32
.sys time:30.53
.Going into lowpower mode. Press user key to wake up
Woke up by GPIO, 5000 ms later into lowpower mode.
sys time:40.201
.sys time:41.231
.sys time:42.252
.sys time:43.273
.sys time:44.295
.Going into lowpower mode. Press user key to wake up
```

1.4.7 WatchDog timer

Watchdog timers are commonly found in embedded systems and other computer-controlled 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 when running untrusted code in a sandbox, to limit the CPU time available to the code and thus prevent some types of denial-of-service attacks.[1] In real-time operating systems, 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

1.5 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.

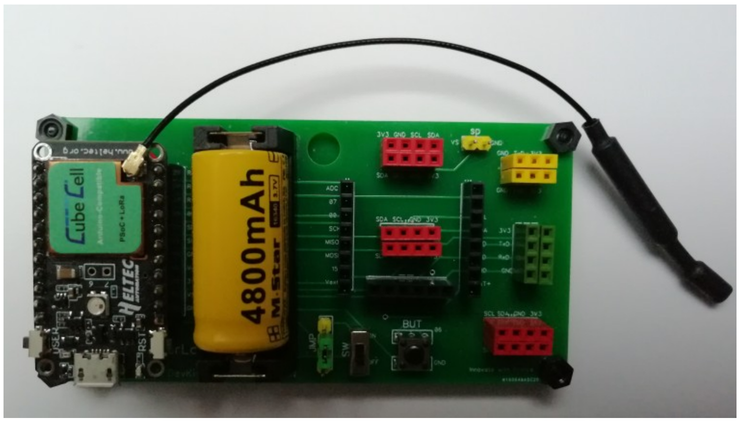


Fig 1.9 Low Power IoT DevKit from SmartComputerLab with CubeCell board

1.5.1 SSD1306 OLED

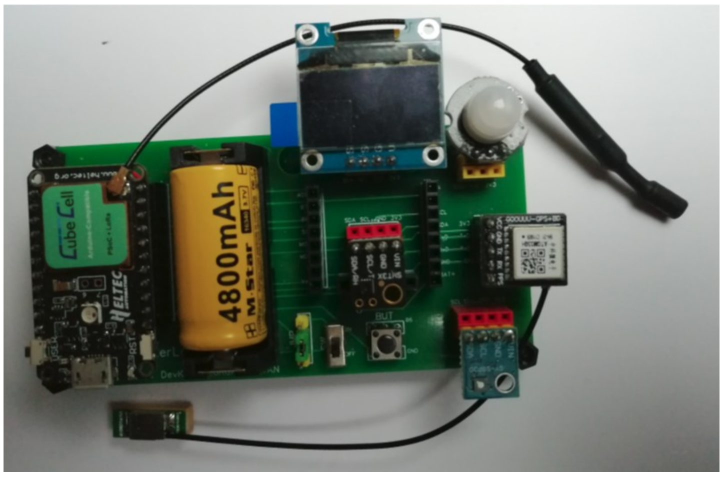


Fig 1.10 IoT DevKit with **OLED** screen connected via **I2C** bus

Note that the 3V3 pin is programmed via:

```
pinMode(Vext, OUTPUT);
digitalWrite(Vext, LOW)
```

`LOW` state means that the outopt (3.3V) is active.

```
#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);
}
```

1.5.2 I2C device scan

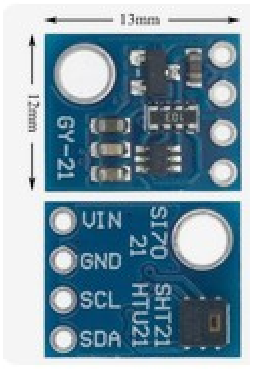
The following code may be used to scan the `I2C` bus to find the connected devices (sensors,displays,..).

```
#include "Arduino.h"
#include "Wire.h"

void setup()
{
  Serial.begin(9600);
  pinMode(Vext,OUTPUT);
  digitalWrite(Vext,LOW);//set vext to high
  Wire.begin(); //(29,28);
}

void loop()
{
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
      Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
  Serial.println("No I2C devices found\n");
  else
  Serial.println("done\n");
  delay(5000);
}
```

The terminal display for SSD1306 OLED on I2C bus (address - `0x3C`)

```
..

Scanning...
I2C device found at address 0x3C  !
done

Scanning...
I2C device found at address 0x3C  !
done

Scanning...
I2C device found at address 0x3C  !
done
```

1.5.3 SHT21 sensor module (I2C)



Driver github:
https://github.com/e-radionicacom/SHT21-Arduino-Library

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()
{
  Serial.begin(9600);
  pinMode(Vext,OUTPUT);
}

int dt, dh;

void loop()
{
  char buff[32];

  digitalWrite(Vext,LOW); // start power before activating Wire
  Wire.begin(29,28);
  SHT21.begin();
  delay(200);
  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();  // end Wire before disconnecting power
  digitalWrite(Vext,HIGH);
  delay(3000);
}
```

Reading SHT21 sensor and display on OLED

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

SHT21 SHT21;

SSD1306Wire  display(0x3c, 100000, SDA, SCL, GEOMETRY_128_64, -1);

void displayOLED(char *line1, char *line2, char *line3)
{
    Serial.println("in oled");
    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();
    delay(1000);
}

float t,h;

void setup()
{
  Serial.begin(9600);
  pinMode(Vext,OUTPUT);
}

int dt, dh;

void loop()
{
  char buff[32],bufft[32],buffh[32];
  digitalWrite(Vext,LOW); //3V3 voltage output activated - Vext ON
  Wire.begin(29,28);
  SHT21.begin();
  delay(200);
  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);
  sprintf(bufft,"T:%d.%d",(int)t,dt);
  sprintf(buffh,"H:%d.%d",(int)h,dh);
  Serial.println(buff);Serial.println(bufft);Serial.println(buffh);
  displayOLED(bufft,buffh," ");
  Wire.end();
  digitalWrite(Vext,HIGH);
  delay(5000);
}
```

1.5.3.1 Simplified SHT21 driver

The following code shows how operates the I2C bus in ordr to activate and to communicate with an SHT21 sensor. Note the control codes and the `Wire` (`Wire1,Wire2`) operations.

```
        #define  eSHT2xAddress     0x40
        #define  eTempHoldCmd      0xE3
        #define  eRHumidityHoldCmd    0xE5
        #define  eTempNoHoldCmd       0xF3
        #define  eRHumidityNoHoldCmd  0xF5

        Wire.beginTransmission(eSHT2xAddress);
        Wire.write(command);
        Wire.endTransmission();
        Wire.requestFrom(eSHT2xAddress, 3);
        Wire.available()
        Wire.read();
```

The complete code:

```
#define  eSHT2xAddress     0x40
#define  eTempHoldCmd      0xE3
#define  eRHumidityHoldCmd    0xE5
#define  eTempNoHoldCmd       0xF3
#define  eRHumidityNoHoldCmd  0xF5
```

```
uint16_t readSensor(uint8_t command)
{
    uint16_t result;
    Wire.beginTransmission(eSHT2xAddress);
    Wire.write(command);
    Wire.endTransmission();
    delay(100);
    Wire.requestFrom(eSHT2xAddress, 3);
    uint32_t timeout = millis() + 300;       // Don't hang here for more than 300ms
    while (Wire.available() < 3) {
        if ((millis() - timeout) > 0) {
            return 0;
        }
    }
    //Store the result
    result = Wire.read() << 8;
    result += Wire.read();
    result &= ~0x0003;   // clear two low bits (status bits)
    //Clear the final byte from the buffer
    Wire.read();
    return result;
}

float sht21_temperature(void)
{
    float value = readSensor(eTempHoldCmd);
    if (value == 0) {
        return -273;                     // Roughly Zero Kelvin indicates an error
    }
    return -46.85 + 175.72 / 65536.0 * value;
}

float sht21_humidity(void)
{
    float value = readSensor(eRHumidityHoldCmd);
    if (value == 0) {
        return 0;                        // Some unrealistic value
    }
    return -6.0 + 125.0 / 65536.0 * value;
}
```

To do:

Test the above examples using both kinds of drivers

Experiment with:

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

and

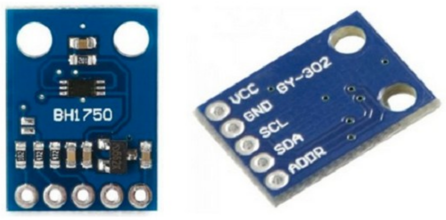
```
digitalWrite(Vext, HIGH); delay(100);
```

Use **PPK2** to analyze the power consumption.

1.5.4 BH1750 – Ambient Light sensor

The BH1750 provides 16-bit light measurements in **lux**, the **SI** unit for measuring light making it easy to compare against other values like references and measurements from other sensors. The BH1750 is able to measure from 0 to **65K+** lux.

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

void setup()
{
  Serial.begin(9600);
  pinMode(Vext,OUTPUT);
}

float lux;
```



```
void loop()
{
  digitalWrite(Vext,LOW); // start power before activating Wire
  Wire.begin(29,28);
  delay(200);
  lightMeter.begin();
  delay(200);
  lux = lightMeter.readLightLevel();
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lux");
  Wire.end();  // end Wire before disconnecting power
  digitalWrite(Vext,HIGH);
  delay(3000);
}
```

1.5.4.1 BH1750 – simplified driver

The following is the BH1759 driver code allowing for direct integration on I2C bus with **`Wire`**, **`Wire1`**, **`Wire2`**, .

- `Wire.beginTransmission(0x23) - address: start of transaction`
- `Wire.write(commande);  - command on one byte`
- `Wire.endTransmission() ;  - end of transaction`
- `Wire.requestFrom( 0x23, 2);   - number of requested bytes (data)`
- `buff = Wire1.read();  - reading one byte`

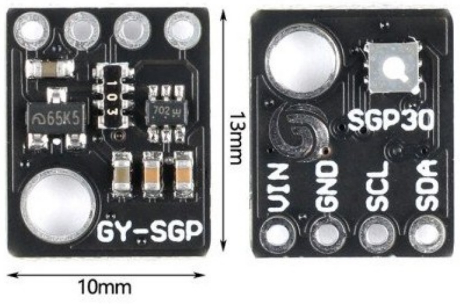
```
#include <Wire.h>

uint16_t readLightLevel(void)
  {
  uint16_t level;
  Wire1.beginTransmission(0x23);
  Wire1.write(0x01);  // power on
  Wire1.endTransmission();delay(50);
  Wire1.beginTransmission(0x23);
  Wire1.write(0x13);  // high resolution - 120 ms, 0x13 - low resolution - 20 ms
  Wire1.endTransmission();
  delay(50);   // high resolution - delay(200);
  Wire1.beginTransmission(0x23);
  Wire1.requestFrom( 0x23, 2);
  level = Wire1.read();
  level <<= 8;
  level |= Wire1.read();
  Wire1.endTransmission();delay(40);
  level = level/1.2; // convert to lux
  return level;
}

void setup()
{
    Serial.begin(9600);
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);delay(50);
}

void loop()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);  delay(50);
    Wire1.begin();
    Serial.print("Luminosity(lux): ");
    Serial.println(readLightLevel());
    Wire1.end();
    digitalWrite(Vext, HIGH);
    delay(1000);
}
```

To do

1. Test the above example
2. Add the display on the OLED screen to show the luminosity value
3. Use **PPK2** to analyze the power consumption.

1.5.5 Air (CO2/TVOC) sensor CCS811 (SGP30)

CCS811 based sensor(I2C) VOC/eCO2 is an air quality monitoring sensor. This sensor is a gas sensor that can detect a wide range of Volatile Organic Compounds (VOCs) and is intended for indoor air quality monitoring. When connected to your micro-controller such as ARS6501 and running corresponding library code it will return a Total Volatile Organic Compound (TVOC) reading and an equivalent carbon dioxide reading (eCO2) over I2C.



The CCS811 has a 'standard' hot-plate MOX sensor, as well as a small micro-controller that controls power to the plate, reads the analog voltage, and provides an I2C interface to read from.
This part will measure eCO2 (equivalent calculated carbon-dioxide) concentration within a range of 400 to 8192 parts per million (ppm), and TVOC (Total Volatile Organic Compound) concentration within a range of 0 to 1187 parts per billion (ppb).

According to the fact sheet it can detect Alcohols, Aldehydes, Ketones, Organic Acids, Amines, Aliphatic and Aromatic Hydrocarbons.

Please note, this sensor, like all VOC/gas sensors, has variability and to get precise measurements you will want to calibrate it against known sources! That said, for general environmental sensors, it will give you a good idea of trends and comparisons.
Also, it is recommended that you run this sensor for 48 hours when you first receive it to "burn it in", and then 20 minutes in the desired mode every time the sensor is in use.
This is because the sensitivity levels of the sensor will change during early use.
The **CCS811** has a configurable interrupt pin that can fire when a conversion is ready and/or when a reading crosses a user-settable threshold.

The CCS811 supports multiple drive modes to take a measurement every 1 second, every 10 seconds, every 60 seconds, or every 250 milliseconds.

The device's I2C address is **`0x5A`**.

```
#include <Wire.h>
#include "Adafruit_SGP30.h"
Adafruit_SGP30 sgp;

uint32_t getAbsoluteHumidity(float temperature, float humidity) {
    // approximation formula from Sensirion SGP30 Driver Integration chapter 3.15
    const float absoluteHumidity = 216.7f * ((humidity / 100.0f) * 6.112f * exp((17.62f *
temperature) / (243.12f + temperature)) / (273.15f + temperature)); // [g/m^3]
    const uint32_t absoluteHumidityScaled = static_cast<uint32_t>(1000.0f * absoluteHumidity);
    return absoluteHumidityScaled;
}

void setup()
{
  Serial.begin(9600);
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW); //set vext to high
  delay(100);
  Wire.begin(29,28);
  Serial.println("SGP30 test");
  while (!sgp.begin()){
    Serial.print(".");delay(500);
  }
  Serial.print("Found SGP30 serial #");
  Serial.print(sgp.serialnumber[0], HEX);
  Serial.print(sgp.serialnumber[1], HEX);
  Serial.println(sgp.serialnumber[2], HEX);
  // If you have a baseline measurement from before you can assign it to start, to 'self-calibrate'
  //sgp.setIAQBaseline(0x8E68, 0x8F41);  // Will vary for each sensor!
}
```

```
int counter = 0;
void loop() {
  if (! sgp.IAQmeasure()) {
    Serial.println("Measurement failed");
    return;
  }
  Serial.print("TVOC "); Serial.print(sgp.TVOC); Serial.print(" ppb\t");
  Serial.print("eCO2 "); Serial.print(sgp.eCO2); Serial.println(" ppm");

  if (! sgp.IAQmeasureRaw()) {
    Serial.println("Raw Measurement failed");
    return;
  }
}
```

To do

- Test the above example
- Add the display on the OLED screen to show the TVOC and CO2 values
- use `digitalWrite(Vext,LOW);` and `digitalWrite(Vext,HIGH );` to set on and off the power line `Vext` in the `loop()` function (**notice the problem !**)

1.5.6 "Time of Flight" distance sensors – VL53L0X and VL53L1X

1.5.6.1 VL53L03

The **VL53L0X** contains a very tiny invisible laser source, and a matching sensor. The **VL53L0X** can detect the "time of flight", or how long the light has taken to bounce back to the sensor. Since it uses a very narrow light source, it is good for determining distance of only the surface directly in front of it. Unlike sonars that bounce ultrasonic waves, the 'cone' of sensing is very narrow. Unlike IR distance sensors that try to measure the amount of light bounced, the VL53L0x is much more precise and doesn't have linearity problems or 'double imaging' where you can't tell if an object is very far or very close.

```
#include "Adafruit_VL53L0X.h"

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

void setup() {
  pinMode(Vext, OUTPUT);
  Serial.begin(9600);
  Serial.println("Adafruit VL53L0X test");
  Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}

void loop() {
  digitalWrite(Vext, LOW);
  delay(50);
  Wire.begin(29,28);
  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    digitalWrite(Vext, HIGH);
    delay(1000);
    return;
  }

  Serial.print("Reading a measurement... ");
  VL53L0X_RangingMeasurementData_t measure;
  lox.rangingTest(&measure, false); // pass in 'true' to get debug data printout!
  Wire.end();
  digitalWrite(Vext, HIGH);
  if (measure.RangeStatus != 4) {  // phase failures have incorrect data
    Serial.print("Distance (mm): ");
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(" out of range ");
  }
  delay(1000);
}
```

1.5.6.2 VL53L1X

This example demonstrates how to read and average distance, the measurement status, and the signal rate.
Use long distance mode and allow up to 50000 us (50 ms) for a measurement.
You can change these settings to adjust the performance of the sensor, but the minimum timing budget is 20 ms for short distance mode and 33 ms for medium and long distance modes.

```
  sensor.setDistanceMode(VL53L1X::Long);
  sensor.setMeasurementTimingBudget(50000);
```

Start continuous readings at a rate of one measurement every 50 ms (the inter-measurement period). This period should be at least as long as the timing budget.

```
  sensor.startContinuous(50);
```

The complete code:

```
#include "Arduino.h"
#include <Wire.h>
#include "VL53L1X.h"

VL53L1X sensor;

void setup(void)
{
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW);
  delay(500);
  Serial.begin(9600);
  Serial.println();Serial.println();delay(100);
  Wire.begin(29,28);
  Wire.setClock(400000); // use 400 kHz I2C
  sensor.setTimeout(500);
  Serial.println("before init");
  if (!sensor.init())
  {
    Serial.println("Failed to detect and initialize sensor!");
    while (1);
  }
  sensor.setDistanceMode(VL53L1X::Long);
  sensor.setMeasurementTimingBudget(50000);
  sensor.startContinuous(50);
}

void loop()
{
  Serial.print(sensor.read());
  if (sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }
  Serial.println();
}
```

To do

- Test the above examples of **VL53LXX** sensors
- Add the display on the OLED screen to show the distance value

1.5.7 `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).

3.5.7.1 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);
}
```

1.5.7.2 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);
}
```

1.5.7.3 UART stream decoded by TinyGPS++ library

```
#include <TinyGPS++.h>
#include <softSerial.h>
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
softSerial ss(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);  // GPIO5 (33) , GPIO3 (8)

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}

static void printFloat(float val, bool valid, int len, int prec)
{
  if (!valid)
  {
    while (len-- > 1) Serial.print('*');  Serial.print(' ');
  }
  else
  {
    Serial.print(val, prec);
    int vi = abs((int)val);
    int flen = prec + (val < 0.0 ? 2 : 1); // . and -
    flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
    for (int i=flen; i<len; ++i) Serial.print(' ');
  }
  smartDelay(0);
}
```

```
static void printInt(unsigned long val, bool valid, int len)
{
  char sz[32] = "*****************";
  if (valid)  sprintf(sz, "%ld", val);
  sz[len] = 0;
  for (int i=strlen(sz); i<len; ++i) sz[i] = ' ';
  if (len > 0)  sz[len-1] = ' ';
  Serial.print(sz);
  smartDelay(0);
}

static void printDateTime(TinyGPSDate &d, TinyGPSTime &t)
{
  if (!d.isValid())  {  Serial.print(F("********** ")); }
  else
  {
    char sz[32];
    sprintf(sz, "%02d/%02d/%02d ", d.month(), d.day(), d.year());
    Serial.print(sz);
  }
  if (!t.isValid())
  {    Serial.print(F("******** "));  }
  else
  {
    char sz[32];
    sprintf(sz, "%02d:%02d:%02d ", t.hour(), t.minute(), t.second());
    Serial.print(sz);
  }
  printInt(d.age(), d.isValid(), 5);
  smartDelay(0);
}

static void printStr(const char *str, int len)
{
  int slen = strlen(str);
  for (int i=0; i<len; ++i)
    Serial.print(i<slen ? str[i] : ' ');
  smartDelay(0);
}

void setup()
{
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW);
  delay(500);

  Serial.begin(9600);
  ss.begin(GPSBaud);

  Serial.println(F("FullExample.ino"));
  Serial.println(F("An extensive example of many interesting TinyGPS++ features"));
  Serial.print(F("Testing TinyGPS++ library v. ")); Serial.println(TinyGPSPlus::libraryVersion());
  Serial.println(F("by Mikal Hart"));
  Serial.println();
  Serial.println(F("Sats HDOP  Latitude   Longitude   Fix  Date       Time     Date Alt    Course
Speed Card  Distance Course Card  Chars Sentences Checksum"));
  Serial.println(F("           (deg)      (deg)       Age                      Age  (m)    --- from
GPS ----  ---- to London  ----  RX    RX        Fail"));

Serial.println(F("-----------------------------------------------------------------------------
---------------------------------------------------"));
}


void loop()
{
  static const double LONDON_LAT = 51.508131, LONDON_LON = -0.128002;

  printInt(gps.satellites.value(), gps.satellites.isValid(), 5);
  printFloat(gps.hdop.hdop(), gps.hdop.isValid(), 6, 1);
  printFloat(gps.location.lat(), gps.location.isValid(), 11, 6);
  printFloat(gps.location.lng(), gps.location.isValid(), 12, 6);
  printInt(gps.location.age(), gps.location.isValid(), 5);
  printDateTime(gps.date, gps.time);
  printFloat(gps.altitude.meters(), gps.altitude.isValid(), 7, 2);
  printFloat(gps.course.deg(), gps.course.isValid(), 7, 2);
  printFloat(gps.speed.kmph(), gps.speed.isValid(), 6, 2);
  printStr(gps.course.isValid() ? TinyGPSPlus::cardinal(gps.course.deg()) : "*** ", 6);

  unsigned long distanceKmToLondon =
    (unsigned long)TinyGPSPlus::distanceBetween(
      gps.location.lat(),
      gps.location.lng(),
      LONDON_LAT,
      LONDON_LON) / 1000;
  printInt(distanceKmToLondon, gps.location.isValid(), 9);

  double courseToLondon =
    TinyGPSPlus::courseTo(
      gps.location.lat(),
      gps.location.lng(),
      LONDON_LAT,
      LONDON_LON);
  printFloat(courseToLondon, gps.location.isValid(), 7, 2);

  const char *cardinalToLondon = TinyGPSPlus::cardinal(courseToLondon);

  printStr(gps.location.isValid() ? cardinalToLondon : "*** ", 6);

  printInt(gps.charsProcessed(), true, 6);
  printInt(gps.sentencesWithFix(), true, 10);
  printInt(gps.failedChecksum(), true, 9);
  Serial.println();
  smartDelay(1000);
  if (millis() > 5000 && gps.charsProcessed() < 10)
    Serial.println(F("No GPS data received: check wiring"));
}
```

Terminal output fragment:

```
..
d    1.0   47.216858  -1.693524   d    07/04/2021 16:49:28 d    34.90  0.50   0.39  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693527   d    07/04/2021 16:49:29 d    34.40  0.50   0.57  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693528   d    07/04/2021 16:49:30 d    34.40  0.50   0.30  N     d          12.77  NNE   d     d          d
d    1.0   47.216858  -1.693528   d    07/04/2021 16:49:31 d    34.00  0.50   0.33  N     d          12.77  NNE   d     d          d
d    1.0   47.216858  -1.693528   d    07/04/2021 16:49:32 d    33.90  0.50   0.41  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693529   d    07/04/2021 16:49:34 d    33.90  0.38   0.24  N     d          12.77  NNE   d     d          d
d    1.0   47.216866  -1.693530   d    07/04/2021 16:49:35 d    32.90  0.38   0.74  N     d          12.77  NNE   d     d          d
d    1.0   47.216866  -1.693527   d    07/04/2021 16:49:36 d    32.50  0.38   0.72  N     d          12.77  NNE   d     d          d
d    1.0   47.216866  -1.693526   d    07/04/2021 16:49:37 d    31.80  0.38   0.61  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693525   d    07/04/2021 16:49:38 d    32.30  0.38   0.59  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693527   d    07/04/2021 16:49:39 d    32.00  0.38   0.13  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693526   d    07/04/2021 16:49:41 d    32.40  0.38   0.61  N     d          12.77  NNE   d     d          d
d    1.0   47.216862  -1.693528   d    07/04/2021 16:49:42 d    32.20  0.38   0.07  N     d          12.77  NNE   d     d          d
d    1.0   47.216858  -1.693526   d    07/04/2021 16:49:43 d    31.80  0.38   0.69  N     d          12.77  NNE   d     d          d
d    1.0   47.216858  -1.693527   d    07/04/2021 16:49:44 d    32.40  0.38   0.22  N     d          12.77  NNE   d     d          d
```

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

1.6 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.

1.6.1 Low power operation with SHT21 sensor

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;
  TimerSetValue( &wakeUp, timetillwakeup ); //timetillwakeup ms later wake up;
  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() {
  Serial.begin(9600);
  Radio.Sleep( );
  TimerInit( &sleep, onSleep ); TimerInit( &wakeUp, onWakeUp );
  onSleep();
}

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

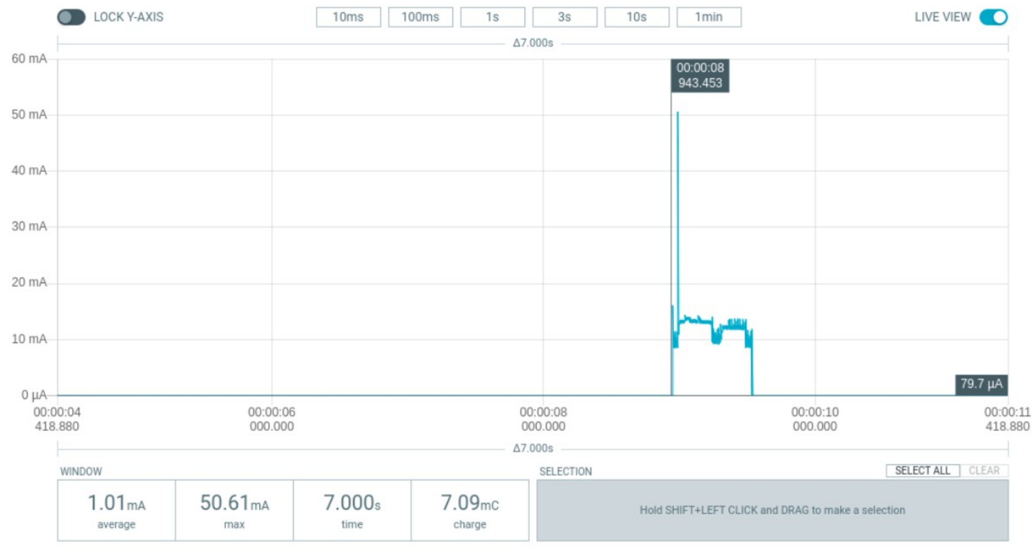


Fig 1.12 The resulting current consumption values from the integrated LiPo battery are:

- **11 mA** for **high power** operations, except the sensor capture (**50 mA**)
- **79.7 µA** for **low power** state

To do

1. Apply the same low/high power operational mode for other sensors.
2. Use simplified SHT21 driver with low precision option (reading time : 10ms)

1.6.2 Direct GPS - 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;
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW);
  softwareSerial.begin(9600);
  //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);
        }
      Wire.end();
      digitalWrite(Vext, HIGH);
      }
    }
  //digitalWrite(Vext, HIGH);
  delay(4000);
}
```

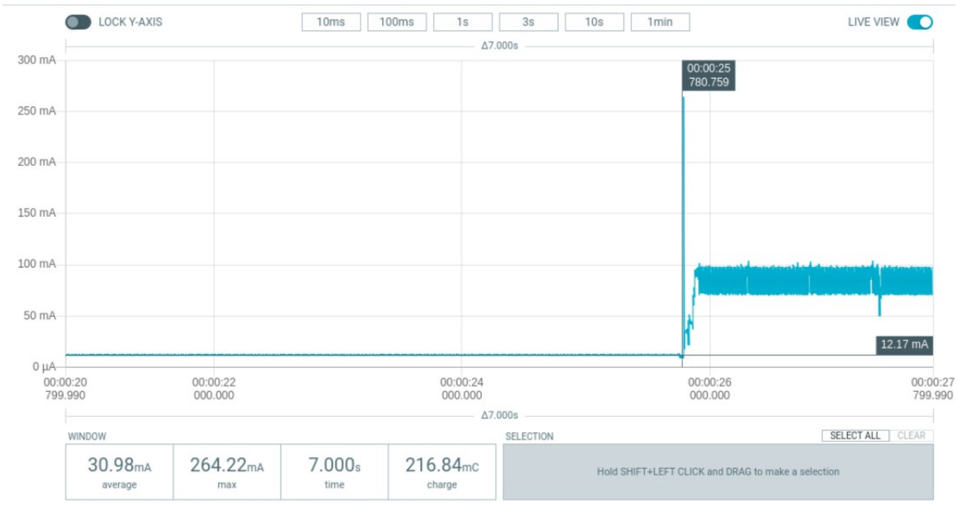


Fig 1.13 GPS read and display with low and high power modes

1.7 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, ..)

1.7.1 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();`

1.7.1.1 Sending data packet (battery voltage)

This first example of LoRa sender implements only some 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; //*2;
      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);
}
```

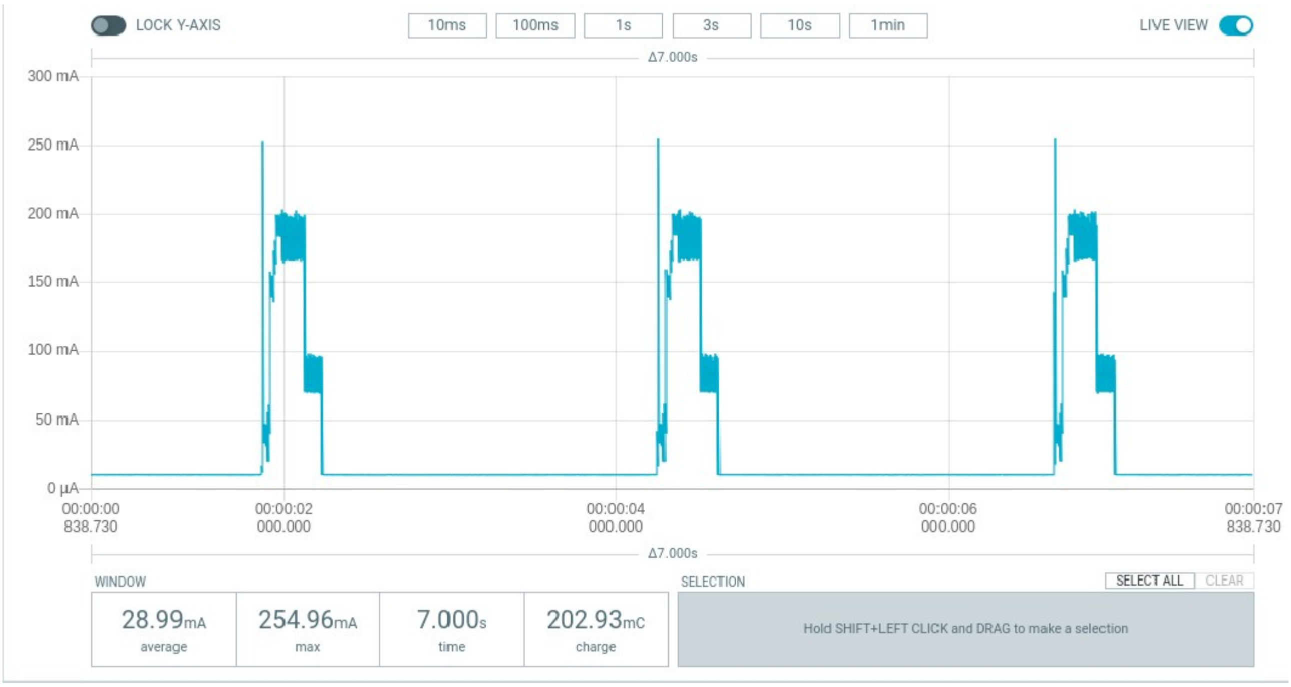


Fig 1.14 Sending three LoRa packets

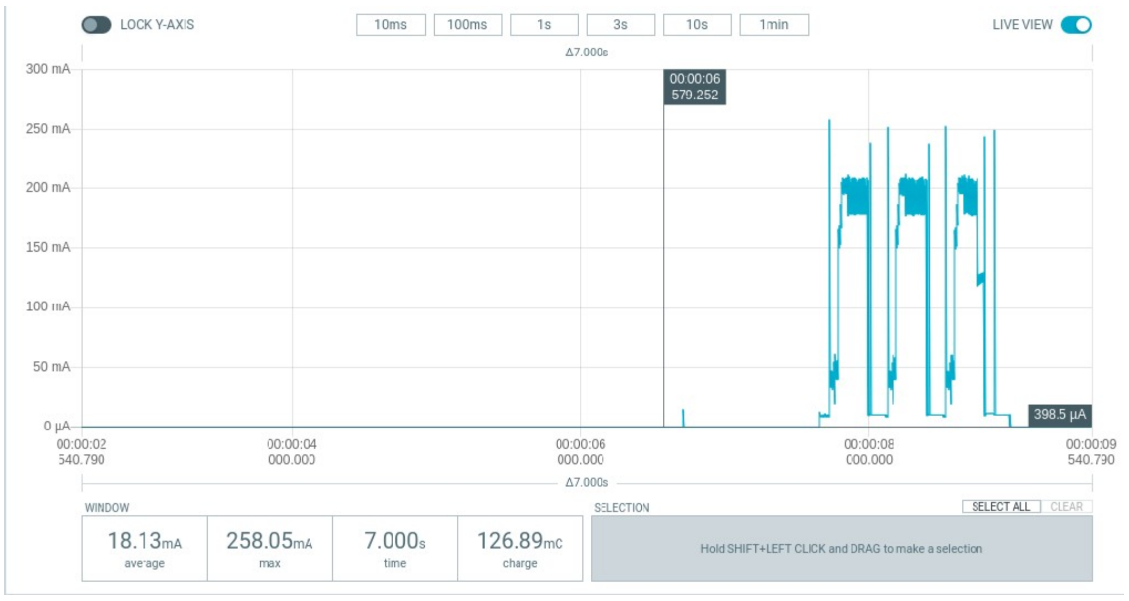
1.7.2 Sending data packet with low_power management

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 **`turnOffRGB()`**.

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.

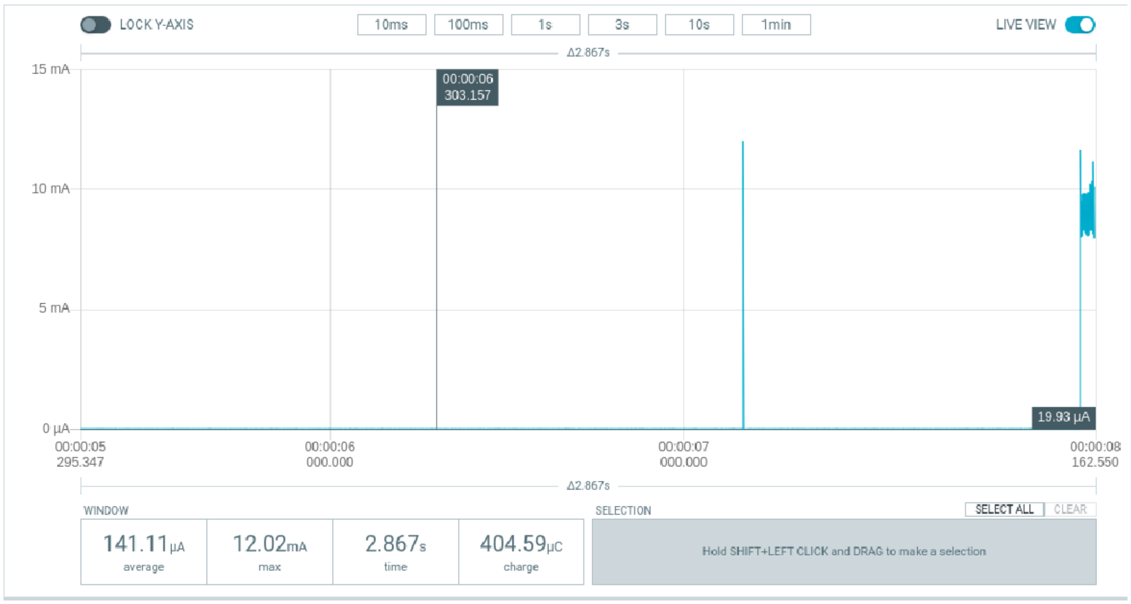
```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#define timetillsleep 1000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif



#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);
}
```



18.13mA average | 258.05mA max | 7.000s time | 126.89mC charge

Hold SHIFT+LEFT CLICK and DRAG to make a selection

Fig 1.15 Sending three LoRa packets



141.11µA average | 12.02mA max | 2.867s time | 404.59µC charge

Hold SHIFT+LEFT CLICK and DRAG to make a selection

Fig 1.16 Sending three LoRa packets in **high_power window** (1000 ms).

Power/current consumption during the **high_power** (up to 200 mA) and **low_power** (**19.93 µA**) states.

1.7.3 LoRa receiver - 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
..
```

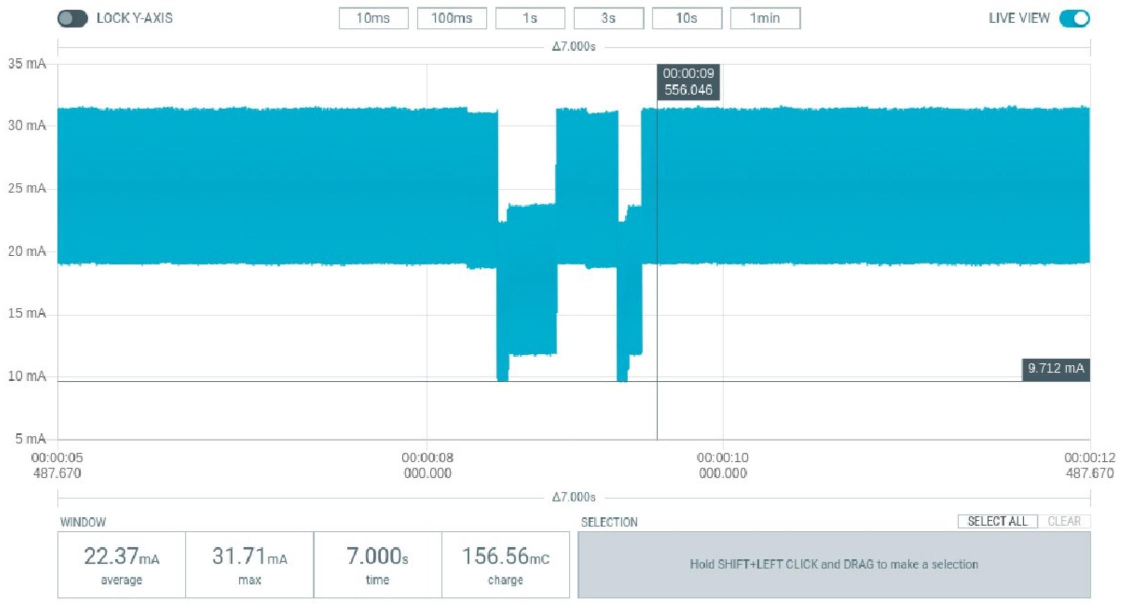


Fig 1.17 Reception of 2 LoRa packets (always active state)

1.7.4 Sending LoRa packets with external 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, CO2, atmospheric pressure, etc.

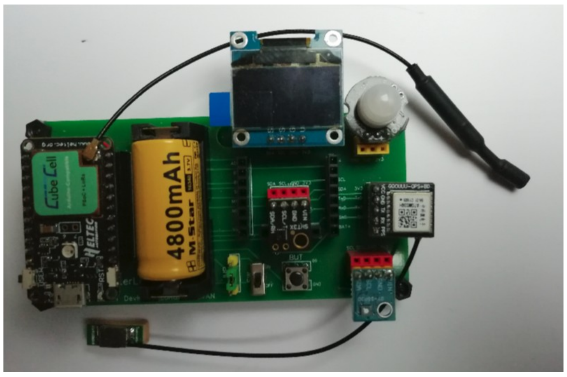


Fig 1.18 IoT CubeCell DevKit with several sensors.

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

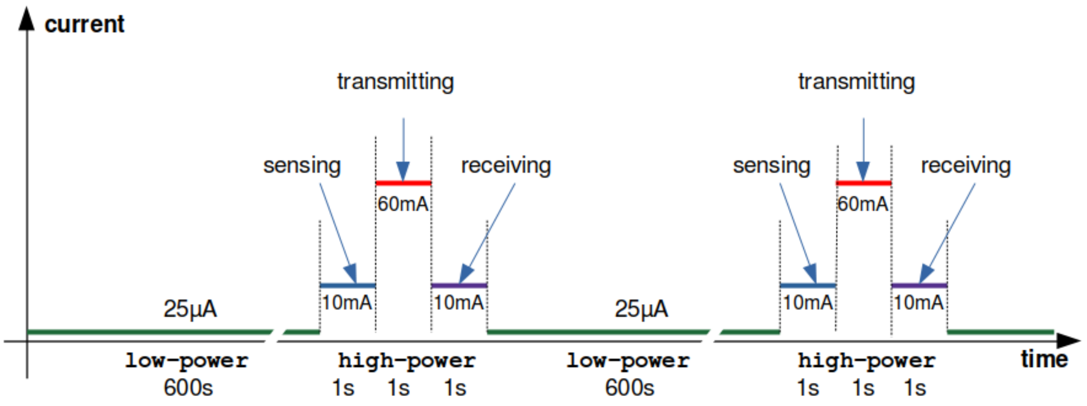


Fig 1.19 Power consumption diagram (current for 3.3V) with low and high power periods and high-power operational phases.

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 1.20 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.

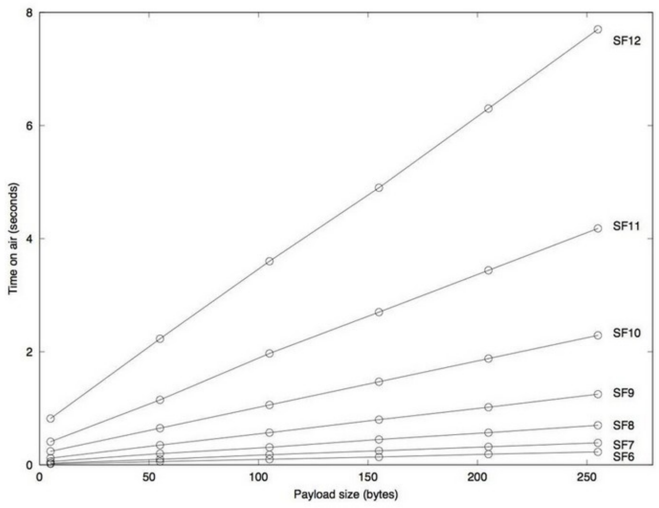


Fig 1.21 Time on Air for different spreading factors and payload size for 125 kHz band and 4/5 coding rate.

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

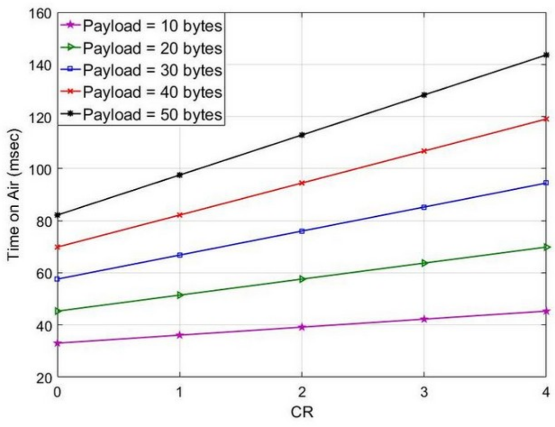


Fig 1.22 Time on Air for different payload size and coding rate (**CR=4/5** to **CR=4/8**).

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.

1.7.5 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).

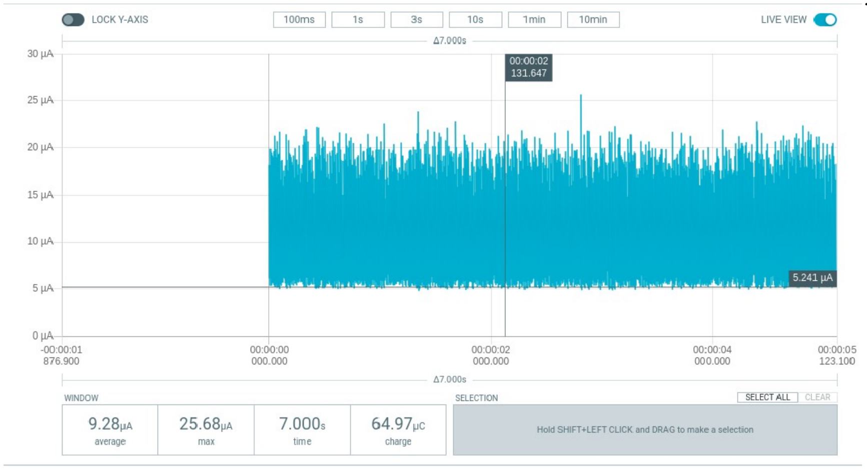
```
      long debut,del=2000;
   // the receiver must respond in this waiting period - reception window
..
      turnOnRGB(COLOR_SEND,0);
      debut=millis();delay(300);turnOffRGB();Radio.Rx(0);
      while(millis()<(debut+del))
        {
        Radio.IrqProcess( ); delay(100);
        }
```

1.7.5.1 The code of terminal node

```
#include "Arduino.h"
#define LoraWan_RGB 5
#include "LoRaWan_APP.h"
#include <Wire.h>
#include "SHT21.h"
SHT21 SHT21;
#define RF_FREQUENCY                              868500000 // Hz
#define TX_OUTPUT_POWER                           14        // dBm - 25mW, 17-50mW, 20 - 100mW
#define LORA_BANDWIDTH                            0         // [0: 125 kHz,
                                                            //  1: 250 kHz,
                                                            //  2: 500 kHz,
                                                            //  3: Reserved]
//  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 );
uint32_t timetillsleep=5000;
uint32_t timetillwakeup=12000;
static TimerEvent_t sleepGo;
static TimerEvent_t wakeUp;
uint8_t lowpower=1, highpower=0;
int16_t txNumber;
bool sleepMode = false;
int16_t rssi,rxSize;
uint16_t voltage;
float t,h;
int dt,dh;
char buff[32];

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

```
void getSHT21()
{
    pinMode(Vext, OUTPUT);
    digitalWrite(Vext, LOW);delay(200);
    Wire.begin(29,28);
    SHT21.begin();
    sdp.sensor[0]=SHT21.getTemperature();
    sdp.sensor[1]=SHT21.getHumidity();
    delay(100);
    Wire.end();
    Serial.println(sdp.sensor[0]);Serial.println(sdp.sensor[1]);
    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; //turnOnRGB(0,0);
  //timetillsleep ms later into lowpower mode;
  TimerSetValue( &sleepGo, timetillsleep );
  TimerStart( &sleepGo );
}

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

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

void setup() {
    Serial.begin(9600);
    pinMode(Vext,OUTPUT);
    digitalWrite(Vext,LOW); //SET POWER
    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( );
  TimerInit( &sleepGo, onSleep );  // activate on event
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}
long debut,del=2000;  // the receiver must respond in this waiting period

void loop() {
  if(lowpower){
    lowPowerHandler();
  }
  if(highpower)
     {
      Serial.println("in highpower");
      getSHT21();
      pinMode(VBAT_ADC_CTL,OUTPUT);
      digitalWrite(VBAT_ADC_CTL,LOW);
      voltage=analogRead(ADC)+400;
      pinMode(VBAT_ADC_CTL, INPUT);
```

```
        sprintf(txPacket,"V:%d,T:%d,H:%d",voltage,(int)sdp.sensor[0],(int)sdp.sensor[1]);
        Serial.printf("sending packet [%s], length: %d\n",txPacket,strlen(txPacket));
        delay(100);
        Radio.Send( (uint8_t *)txPacket, strlen(txPacket) );
        highpower=0;lowpower=1;
      }
 Radio.IrqProcess( );
                                                                                          }
```



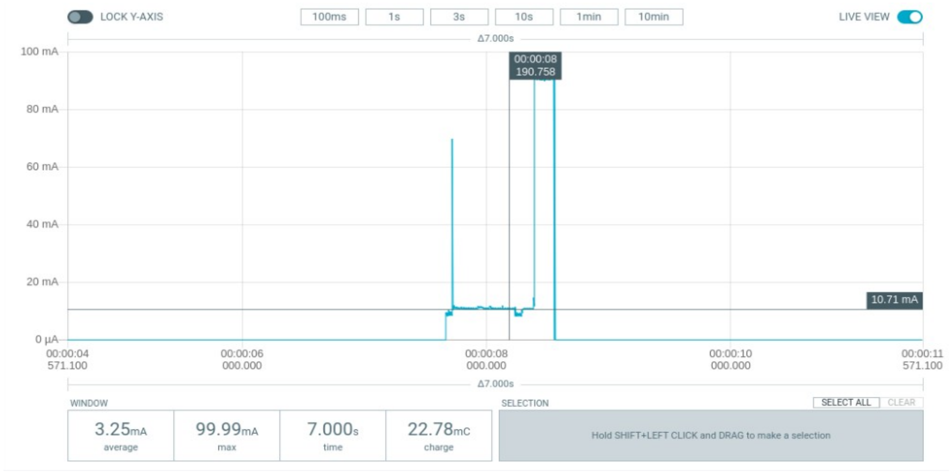


Fig 1.23 (a) (b) Low_power and high_power sensing periods

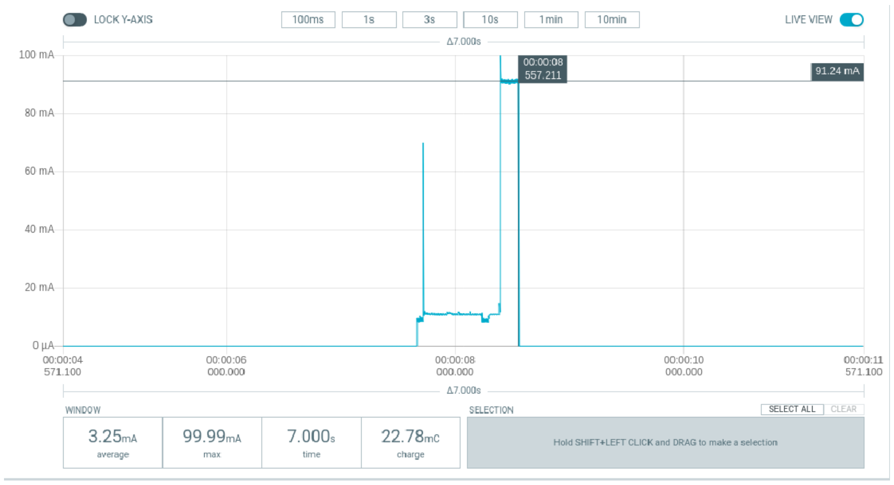


Fig 1.23 (c) high_power transmission period

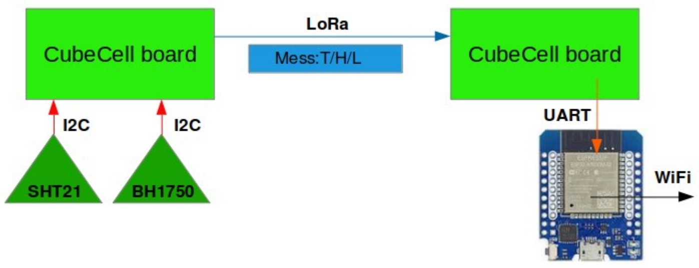
```
..
in highpower
21.23
35.37
sending packet [V:4028,T:21,H:35], length: 16
TX Going into lowpower mode, 12000 ms later w?Woke up, 5000 ms later into lowpower mode.
in highpower
21.20
35.44
sending packet [V:4046,T:21,H:35], length: 16
TX Going into lowpower mode, 12000 ms later w?Woke up, 5000 ms later into lowpower mode.
in highpower
21.17
35.37
sending packet [V:4046,T:21,H:35], length: 16
TX Going into lowpower mode, 12000 ms later w?Woke up, 5000 ms later into lowpower mode.
in highpower
21.18
37.14
sending packet [V:4046,T:21,H:37], length: 16
TX Going into lowpower mode, 12000 ms later w?Woke up, 5000 ms later into lowpower mode.
in highpower
..
```

To do:

- Test the above codes
- Modify the radio parameters such as **spreading factor**, **signal bandwidth** and the **transmission power** and observe the current consumption

1.8 LoRa-WiFi Gateways for CubeCell terminals

In this lab we are going to introduce the complete IoT architectures including terminal nodes and the gateway nodes allowing us to communicate with the IoT servers.
The simplest way to extend the capacity of the receiver node is to add a **WiFi modem**. This can be done by adding an ESP32 or ESP8266 board to our kit and connect both MCU**s** via an UART link.



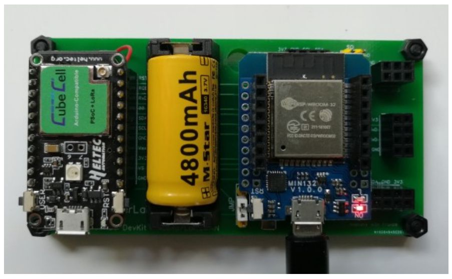


Fig 1.23 Simple LoRa-WiFi gateway with ESP32 (Wemos mini D1) and **UART** bus

1.8.1 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.

1.8.1.1 Sending data from CubeCell via an UART link

```
// this code works on CubeCell Board (HTCC-AB01)
#include "softSerial.h"
softSerial ss(GPIO5 /*TX pin*/, GPIO3 /*RX pin*/);

void setup()
{
  Serial.begin(115200);delay(200);
  Serial.println();
  Serial.println("softSerial init");
  ss.begin(9600);delay(200);
}

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(1000);
}
```

1.8.1.2 Receiving data on ESP32 via UART from CubeCell

```
// this code works on WEMOS D1 MINI ESP32 board
#include <Arduino.h>
HardwareSerial uart(2); // 16 -RX,  17-TX

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(115200);
    delay(200);
  Serial.println();
  Serial.println("Start UART on 16-RX and 17-TX");
  // set the data rate for the SoftwareSerial port
  uart.begin(9600);  delay(2000);
}

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

void loop()
{
if (uart.available())
  {
  if(i==0) Serial.println("Receiving on UART from CubeCell");
  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>15000)  //the data may be sent every 20 seconds
      {
      Serial.println("You can put here ThingSpeak update");
      Serial.println("Fields update");
      lasttime=millis();
      }
    }
  }
  else
    {
    delay(1000);
    i=0;
  }
}
```

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

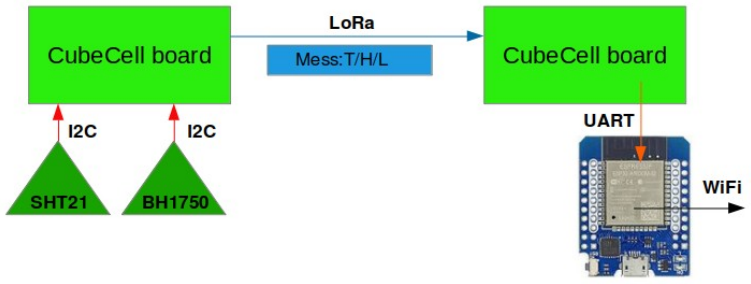
On the ESP32 board side we use `HardwareSerial`. The board receives the data on the `HardwareSerial` link and sends them to the ThingSpeak server. The **ESP32** board is connected to ThingSpeak server via a WiFi link. The received (via `uart`) data are sent to the ThingSpeak server every 20 seconds.

```
// this code works on WEMOS D1 MINI ESP32 board
#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;
  }
}
```

1.8.2 LoRa sender and 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 .**



1.8.2.1 CubeCell low power LoRa sender

```
#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......");
}
```

1.8.2.2 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

1.9 ESP32 LoRa receiver and WiFi gateway node to ThingSpeak

In the following example we use a complete **ESP32 IoT DevKit** with ESP32 and LoRa modem to receive/send the Lora packets and to relay them to ThingSpeak server.
The use of an independent node with complete set of interfaces allows us to build more "intelligent" gateway than in previous case.
As we can see this node integrates Heltec ESP32 WiFi-LoRa board and a set of bus interfaces. With these buses we may connect a number of peripheral devices such as **external EEPROM** modules or m**icro-keyboards**, RFID card scanners, and many more.
These devices allow us to provide the identifiers and codes for WiFi management and for ThingSpeak or MQTT server channels and access codes.

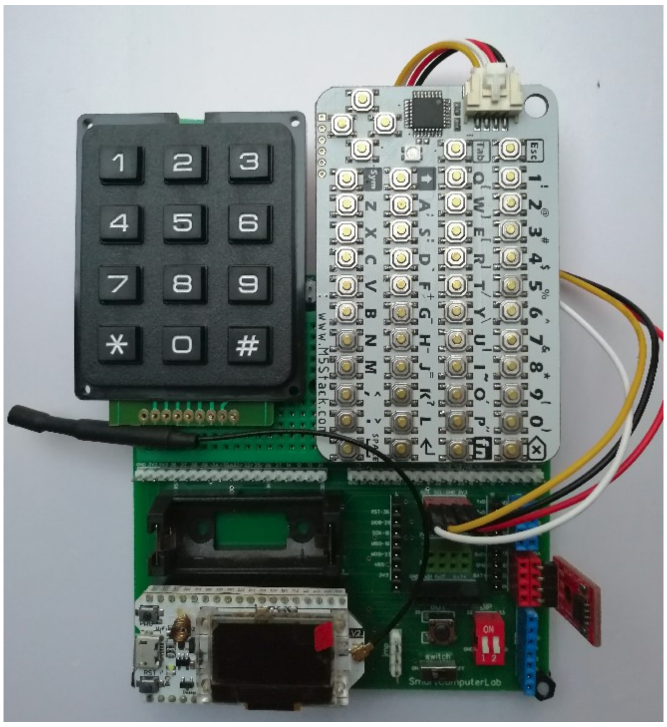


Fig 1.24 Pomme-Pi Heltec ESP32 IoT DevKit board with LoRa modem and external keyboards and EEPROM module.

The following is a simple code, without external parameters, running on ESP32:

```
#include <SPI.h>
#include <LoRa.h>
#include <WiFi.h>
#include "ThingSpeak.h"
char* ssid     = "PhoneAP";
char* pass  = "smartcomputerlab";
WiFiClient client;
unsigned long myChannelNumber = 114;   // Thinspeak channel
const char *myWriteAPIKey="E8KYBCRD2Z59LVWJ" ;  // write code

#define SCK     18   // GPIO18 -- SX127x's SCK
#define MISO    19   // GPIO19 -- SX127x's MISO
#define MOSI    23   // GPIO23 -- SX127x's MOSI
#define SS       5   // GPIO05 -- SX127x's CS
#define RST     15   // GPIO15 -- SX127x's RESET
#define DI0     26   // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq    8685E5
#define sf      9
#define sb      125E3

typedef union
{
  uint8_t frame[16]; // trames avec octets
  float  data[4];    // 4 valeurs en virgule flottante
} pack_t ;  // paquet d'émission
```

```
QueueHandle_t dqueue;  // queues for data packets

void onReceive(int packetSize)
{
pack_t rdp;  int i=0;
if (packetSize == 0) return;   // if there's no packet, return
if (packetSize==16)
  {
  while (LoRa.available())  {  rdp.frame[i]=LoRa.read();i++; }
  xQueueReset(dqueue); // to keep only the last element
  xQueueSend(dqueue, &rdp, portMAX_DELAY);
  }
}

void setup()
{
  Serial.begin(9600);
  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); delay(1000);
  Serial.println("ThingSpeak begin");
  Serial.println("Start Lora");
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DI0);
  Serial.println();delay(100);Serial.println();
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
LoRa.setCodingRate4(5);
LoRa.setPreambleLength(8);
dqueue = xQueueCreate(4,16); // queue for 4 data packets
LoRa.onReceive(onReceive);  // pour indiquer la fonction ISR
LoRa.receive();             // pour activer l'interruption (une fois)
}

uint32_t mindel=10000;  // 10 seconds

union
{
  char cbuff[32];
  uint8_t bbuff[32];
} sdp;

int count=20000;

void loop()
{
pack_t rp;    // packet elements to send
  xQueueReceive(dqueue,rp.frame,portMAX_DELAY); // 6s,default:portMAX_DELAY
  ThingSpeak.setField(1,rp.data[0]);
  ThingSpeak.setField(2,rp.data[1]);
  ThingSpeak.setField(3,rp.data[2]);
  ThingSpeak.setField(4,rp.data[3]);
  Serial.printf("d1=%2.2f,d2=%2.2f,d3=%2.2f,d4=%2.2f\n",rp.data[0],rp.data[1],rp.data[2],
               rp.data[3]);
  while (WiFi.status() != WL_CONNECTED) {  delay(500);  }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  count=10000 + random(10000);sprintf(sdp.cbuff,"%d",count);
  delay(400);
  LoRa.beginPacket();
  LoRa.write(sdp.bbuff,strlen(sdp.cbuff));
  LoRa.endPacket();
  delay(mindel);  // mindel is the minimum waiting time before sending to ThingSpeak
  LoRa.receive();
}
```

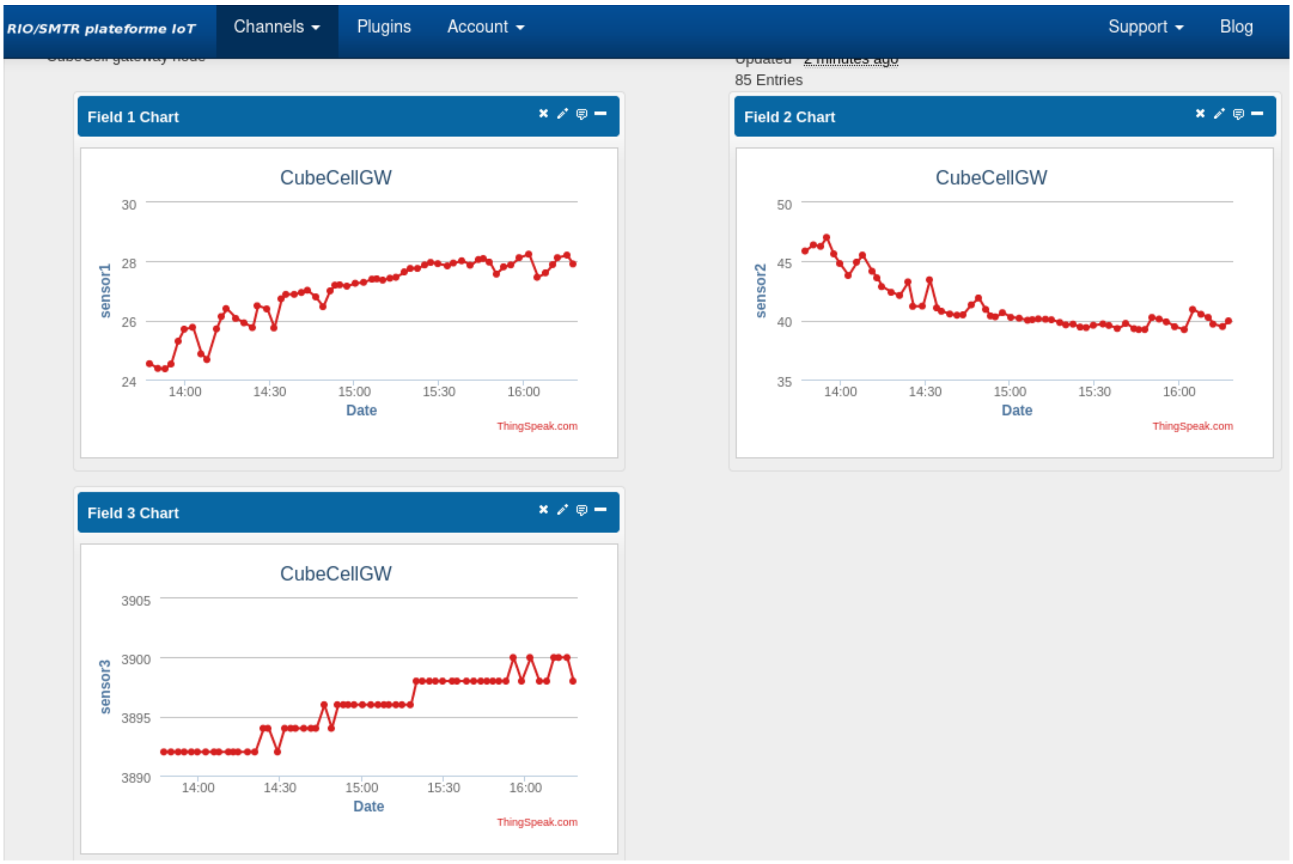


Fig 1.25 ThingSpeak charts - Temperature, humidity and battery level (node with solar cell).

Note that the battery level is going up during the day-light period.

To do:

- Use SmartConfig and ESPTouch application (Android) to provide network credentials to your ESP32 board
- Use the CardKB micro-keyboard to provide network credentials to your ESP32 board
- Add an external EEPROM to prepare the LoRa parameters. Write a program to store WiFi and ThingSpeak parameters on the EEPROM module. Modify the CubeCell sender in order to read the LoRa parameters form the external EEPROM if it is available.
- Modify the data packet by adding the Chip-ID value (6-bytes) – node address. The gateway node sends the ACK packet with the terminal node address. The terminal node ignores the packets with the address not corresponding to its Chip-ID value.
- Build a gateway sending the data messages to an MQTT broker (instead of ThingSpeak server).

Some useful utilities and drivers:

An example of code for **SmartConfig** for a 2.4GHz WiFi network.

```
#include "WiFi.h"

void setup() {
  Serial.begin(115200);

  //Init WiFi as Station, start SmartConfig
  WiFi.mode(WIFI_AP_STA);
  WiFi.beginSmartConfig();

  //Wait for SmartConfig packet from mobile
  Serial.println("Waiting for SmartConfig.");
  while (!WiFi.smartConfigDone()) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("SmartConfig received.");

  //Wait for WiFi to connect to AP
  Serial.println("Waiting for WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("WiFi Connected.");

  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

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

}
```

An example code to read the KBCard keyboard connected to I2C bus:

```
#include <Wire.h>
#include <U8x8lib.h>  // bibliothèque à charger a partir de
// the OLED used
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);

#define CARDKB_ADDR 0x5F

int val=99;
char buff[32];

char ssid[32],pass[32];
int i=0;

char cbuff[256]; int line=1;

int set_wifi(char *s)
{
 memset(s,0x00,32);
 while(1)
 {
  Wire.requestFrom(CARDKB_ADDR, 1);
  while(Wire.available())
  {
    char c = Wire.read(); // receive a byte as characterif
    if (c != 0)
    {
      Serial.println(c, HEX);
      Serial.println(c);
      if(c=='\n' || c=='\r')  // line feed or carriage return
        { u8x8.clear(); i=0; return(0);}
      else
        { s[i]=c; u8x8.drawString(0,i/16+line,s+16*(i/16));i++; }
      if(i==32)
        { u8x8.clear(); i=0; memset(s,0x00,32); return(1); }
    }

  }
   delay(10);
 }
}


void setup()
{

  Serial.begin(9600);
  Wire.begin(21,22);  // CardKB connected to I2C bus
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  sprintf(buff,"%d", val);
  u8x8.drawString(0,0,"set WiFi");
  delay(3000);
  u8x8.drawString(0,0,"set SSID");
  set_wifi(ssid);
  Serial.println(ssid);
  u8x8.drawString(0,0,"set PASS");
  set_wifi(pass);
  Serial.println(pass);
  u8x8.clear();
}


void loop()
{
  Wire.requestFrom(CARDKB_ADDR, 1);
  while(Wire.available())
  {
    char c = Wire.read(); // receive a byte as characterif
    if (c != 0)
    {
      Serial.println(c, HEX);
      Serial.println(c);
      if(c=='\n' || c=='\r')  // line feed or carriage return
        { u8x8.clear(); i=0; memset(cbuff,0x00,256); }
      else
        { cbuff[i]=c; u8x8.drawString(0,i/16,cbuff+16*(i/16));i++; }
      if(i==256)
        { u8x8.clear(); i=0; memset(cbuff,0x00,256); }
    }

  }
   delay(10);
}
```

Lab 2: (Low Power IoT)

Building LoRaWAN applications with TTN/TTS

NOTE 28th June 2021: The Things Network (TTN) is moving officially to The Things Stack (TTS). TTN is no longer available to create LoRaWAN gateways, so we've adjusted this guide to reflect this change.

2.1 Introduction - What is LoRaWAN ?

LoRaWAN is a radio communication protocol **based on LoRa technology**. In the context of the Internet of Things, it makes it possible to build a **low-power-consumption wide-area network** (**LPWAN**), integrating terminal equipment with low power consumption via gateways.
LoRaWAN protocol is used in the context of smart cities, industrial monitoring or even agriculture. It is based on proprietary modulation technology LoRa, created in 2009 by the Grenoble start-up Cycléo (acquired by Semtech in 2012). **Semtech** promotes LoRa through the **LoRa Alliance**, of which it is a founding member.
LoRa is the physical layer allowing to connect sensors or objects requiring a long battery life (counted in years), in a volume and with a reduced cost.
LoRaWAN stands for **long-range wide-area network**.

2.1.1 General description

The **LoRaWAN** protocol **is a communication protocol** for the **Internet of Things** that uses a proprietary **spread spectrum modulation** technique (**chirp spread spectrum**) called LoRa. This protocol aims to be simple, inexpensive to implement and energy efficient.

LoRaWAN targets **low-cost, low-power long-range** communications, rather than high-speed communications (which consume **more energy** and **computing power**).
The challenges concerning the interconnection of objects reside in the cost of the latter, their autonomy as well as their number from a network point of view.
This **low cost** is obtained by the use of a **star architecture** (easier to implement than a mesh architecture), a modulation technique that is easier to implement than that of conventional cellular networks (reducing the cost of electronic components dedicated) as well as the use of **free frequency bands** (usable without charge).

LoRaWAN terminal equipment is mostly inexpensive in order to be able to be deployed in large numbers.
The use of free frequencies also allows the creation of so-called **private LoRa networks** (independent of operators), which can cover a building such as a field or a city **without subscription costs**.

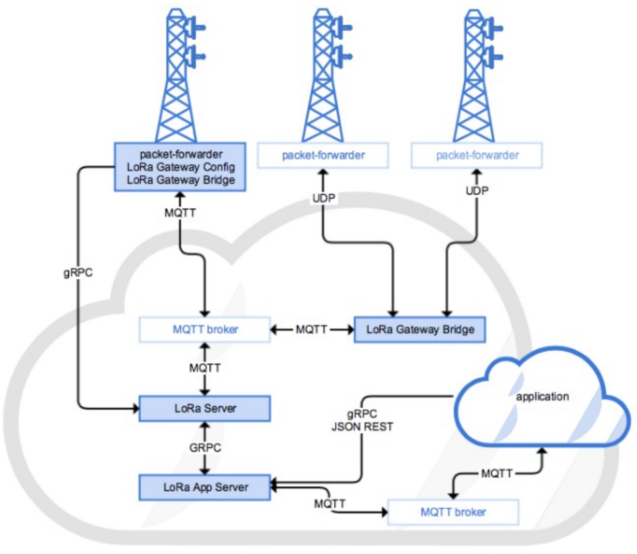


Fig 2.1 Diagram illustrating a **LoRaWAN architecture** with the antennas capturing the LoRa radio links from the terminals, packet-forwarders, network and application servers.

A LoRaWAN network consists of **low-power wireless devices** (Terminals-T) that communicate with **application servers (AS) through gateways (G)**. The modulation technique used between equipment and gateways is LoRa.

The communication between the **gateways and the servers** is established via the **IP protocol** by means of an Ethernet/WiFi or cellular network.

The LoRaWAN network **topology** is said to be **star-of-stars** because an application server is connected to a multitude of gateways which are themselves connected to a multitude of terminal equipment.

In the network sense, the devices are not connected to the **gateways**, they serve only as a **relay** (**packet forwarder**) to reach the **server managing the network**, itself connected to one or more application servers.

The packets sent by the devices are re-transmitted by the gateways (packet-forwarders) after adding only information concerning the **quality of the received signal** (RSSI, SNR,..)

If the radio coverage allows it, several gateways can re-transmit the same message from a device, it is then duplicated in the collection network, it is the server hosting the application which ensures the splitting of the packets.
This allows in particular the **localization of equipment** via the **comparison of the different arrival times** for the same duplicated packet.

When a response must be sent by the server, it uses the information added by the gateways concerning the signal quality in order to choose which one to send the response packet to.

LoRaWAN does not allow direct dialogue between two connected objects. If such a dialogue must take place, it is done through the application server.

2.1.2 Modulation LoRa

Lora uses a proprietary **chirp spread spectrum modulation called LoRa**. This modulation is mainly done on the **ISM 868** (MHz) radio bands in Europe and 915 in North America. The use of **CSS modulation** for the Internet of Things has been patented by **Cycléo**, a French company that was acquired by **Semtech** in 2012.
This modulation allows on average a distance between a gateway and a device of up to 5 Km in urban areas and 15 Km in rural areas.
Spread spectrum modulation techniques such as LoRa use a larger bandwidth than what is ideally necessary for a given bit rate but take advantage of this frequency spread to operate with a **weak or highly noisy** signal.
Spreading the spectrum consists in **repeating several times** the message transmitted at different frequencies.
The **frequency variation** performed by LoRa **is linear**, which allows receivers to simply eliminate frequency shifts and Doppler effects inherent in signal transmission. This operation allows LoRa transmitters to be produced at low cost.
This spread spectrum technique also allows the sensors to be less sensitive to the Doppler effect and therefore to more easily transmit **messages sent on the move** (on a moving TGV for example).
The **spreading factor - SF** is generally **set by the server** when a terminal equipment is connected to the network, by sending a **request to measure the signal-to-noise ratio**.
LoRa defines the spread spectrum factor (SF) by the formula:

$$\mathbf{SF = \log_2(Rc/Rs)}$$

with **Rc** being the rate of the transmitted
message and **Rs** the rate of the symbol to be transmitted.

Increasing the Spreading Factor makes it possible to cover a greater distance between the equipment and the gateway **to the detriment of the available bandwidth**.

The different SFs are **orthogonal**, which means that **several frames can be received at the same time** on the same channel provided that they use a **different SF**. If two frames are received at the same time by a gateway with a difference of less than 6dB on the same channel and with the same SF, they will be lost because they cannot be differentiated.

The possible bandwidths to configure for a channel are **125**, **250** and **500** KHz for the **868** band, which makes it possible to reach a maximum **rate** of **22 kbit/s** with a **bandwidth** of **500** KHz and a **Spreading Factor** of **7.**

Data Rate (DR)	Modulation	Spreading Factor (SF)	Bandwidth	datarate (bit/s)

0	LoRa	SF12	125	250
1	LoRa	SF11	125	440
2	LoRa	SF10	125	980
3	LoRa	SF9	125	1 760
4	LoRa	SF8	125	3 125
5	LoRa	SF7	125	5 470
6	LoRa	SF7	250	11 000
7	FSK	50kbit/s		50 000
8	for uture utilization			

The use of free frequencies requires respecting a **maximum occupation time** of the radio channel (**duty-cycle**). Maximum channel occupancy is **1%** in Europe on the 868 MHz band. On the 868 band, the LoRa specification **initially mandates 3 channels** with a width of **125 KHz** common to all **868.10**, **868.30** and **868.50** so that the activation message can be received by the server.

This parameter can then be modified by the network and allow the sensor to distribute its messages over a greater number of channels. The equipment can therefore randomly distribute its transmissions over each of these bands while respecting the regulatory occupation time.

LoRa allows you to set the main radio parameters using the **Data Rate parameter**. The Data Rate is defined by a value from **0 to 15** and sets the **type of modulation**, the **spreading factor** as well as the **bandwidth** used.

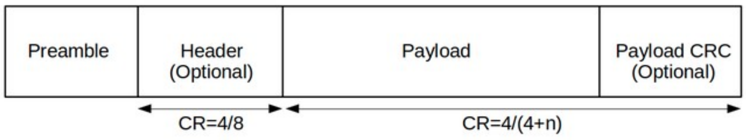


Fig 2.2 LoRa Physical Frame

The LoRa physical frame consists of a **preamble**, a **header**, the **data** and then an **error check**.

- The **header** is present in the default transmission mode (explicit), it is transmitted with a **code rate** of **4/8**. It indicates the **data size**, the **code rate** for the rest of the frame and it also indicates if a **CRC is present**. A CRC is present in the header to allow the receiver to detect if it is corrupted.
- The **maximum data size is between 51 and 222** bytes **depending** on the **spreading factor used** (the larger the SF the smaller the data size).
- The **code rate** for the rest of the frame can be set from **4/5** to **4/8**, the **4/5** rate being the most used.

According to these parameters, it is therefore possible to define the useful flow rate **Rb** by the mathematical formula:

$$\mathbf{Rb = SF * BW/(2^{SF}) * CR}$$

2.1.3 The LoRaWAN protocol

LoRaWAN is a **media access control** type protocol. Its operation is simpler than that of cellular technologies which rely on powerful and therefore more expensive terminal equipment than those used in the Internet of Things. The protocol is not symmetrical, and there are differences between the **uplink** messages coming from the objects and the **downlink** messages coming from the application and intended for the objects.

It is based on an **ALOHA type operation** for sending messages, so when a device has to send data it does so without checking if the channel it is going to use is available, and repeats this sending on different channels without knowing if this message was correctly received.
An acknowledgment message can be requested by the object, which can automatically repeat sending if this message has not been received.

Downlink messages sent by the gateway have a higher network cost, because if the gateway sends a message, it can no longer listen to the other messages sent during this time.
The protocol defines **3 classes of equipment** (**A**, **B** and **C**). **Class A must be implemented in all equipment** for compatibility. Equipment can change class during operation.

Class A: This class has the **lowest energy consumption**. When the equipment has data to **send**, it does so without checking then it opens **successive listening windows** for any messages coming from the server, the recommended durations are **1 then 2 seconds**. These 2 windows are the only ones during which the server can send the equipment the data it has previously stored for it.

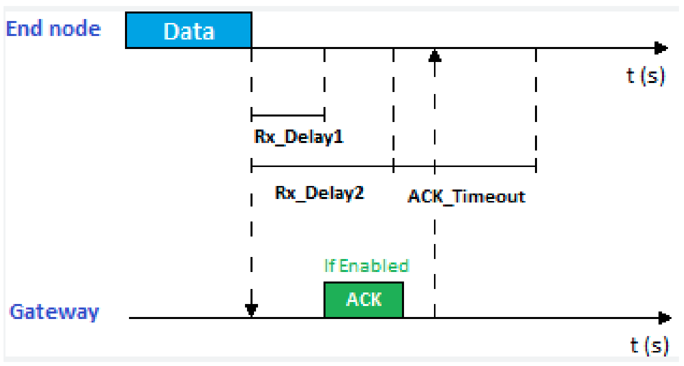


Fig 2.3 Operational phases of **LoRaWAN Class A** protocol

Class B: This class allows a compromise between power consumption and the need for two-way communication. These devices open **reception windows** at **intervals programmed** by periodic messages sent **by the server**.

Class C: This class has the **highest power consumption** but allows **bi-directional communications** that are **unscheduled**. The equipment has a **permanent listening window**.

The format of LoRaWAN packets is described in the diagram below. Field **sizes** are given in **bits**.

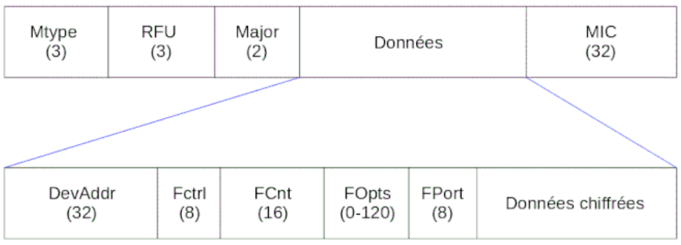


Fig 2.4 LoRaWAN packet structure

Here is the definition of the different fields contained in a LoRaWAN packet:

- **`Mtype`**
  This field indicates the type of message (up or down).
- **`RFU`**
  This field is reserved for future use.
- **`Major`**
  This field indicates the version of the protocol used.
- **`MIC`**
  This field allows the integrity calculation of the packet in order to detect if it has been altered during its transport.

- **`DevAddr`**
  This field contains the device address.
- **`FCtrl`**
  This field allows bit rate adaptation and acknowledgments. It indicates the presence of additional packets as well as the length of the **`FOpts`** field.
- **`FCnt`**
  This field is a frame counter (increment with each sending).
- **`Points`**
  This field is used to pass MAC commands (connectivity control by a device for example).
- **`FPort`**
  This field contains the port of the application or service to which the packet is addressed.

In order to be able to operate on the network, a device must have been **activated**. **Two activation procedures** are possible:

- **Activation By Personalization** (**ABP**): **Encryption keys are stored in the devices**;
- **Over The Air** (**OTAA**): **Encryption keys are obtained by exchange with the network**.

The table below summarizes the information transmitted by the end device during the activation procedure.

ID	Property	Obtaining
`devAddr`	End device identity (32bits)	**Generated in OTAA**, configured in ABP
`devEUI`	Endpoint Identity (64bit)	**Configured**
`appEUI`	Identity of the application (makes the owner of the endpoint unique)	**Configured**
`nwkSKey`	Key used by server and endpoint to calculate and verify MIC field	**Generated in OTAA**, configured in ABP
`appSKey`	Key used by server and endpoint to encrypt and decrypt packet data Generated in OTAA, configured in ABP	**Generated in OTAA**, configured in ABP
`appKey`	Key used by the end device during the OTAA procedure Generated in OTAA, configured in ABP	**Generated in OTAA**, configured in ABP

Note that with **OTAA** scheme there are only two parameters to be configured during the programming/flashing phase:

```
devEUI and appEUI
```

The **`FOpts`** field present in LoRaWAN packets allows devices to send network commands.
The **`LinkCheckReq`** command allows an end device to test its connectivity.
The rest of the commands are used by the server to set the radio parameters of the terminal equipment such as the **Data Rate** or the channel for example. Other commands allow you to control their **battery level** or their **reception quality**.

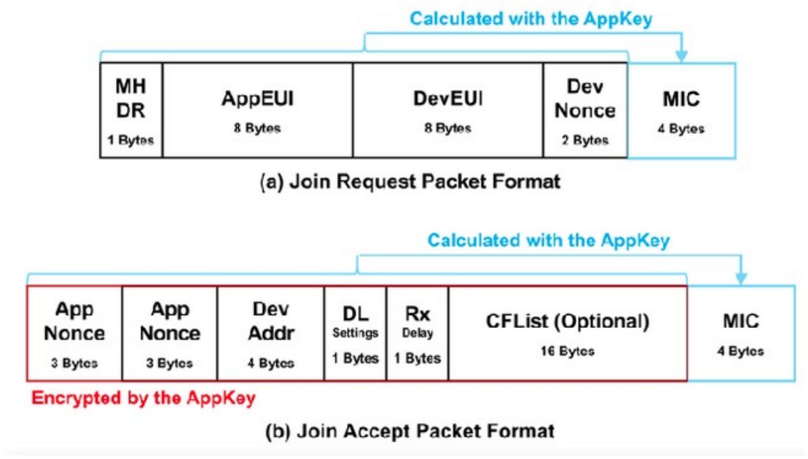


Fig 2.5 Join Request and **Join Accept** packets: sending **AppEUI** and **DevEUI** (non encrypted), receiving **DevAddr** (32 bits - encrypted)

2.1.4 Energy consumption

LoRaWAN consumes little energy. It uses a simplified version of the ALOHA protocol for the MAC layer, a star topology, cyclic transmission in each sub-band and has defined three classes of end devices to shift the complexity as much as possible to the base station.

- The power optimization parameters used for the transmit-receive cycle are based on the use of **modes** (**transmit**, **receive**, **standby**, and **sleep**), information transmission strategy, and transmit power.

The **power consumption of objects** (end devices) in the LoraWan network is based on the use of the four main modes (transmit, receive, standby and sleep) and the **time spent in each mode**.

The comparison is made from the consumption of the components (for LoRaWan RN2903, LoRaWAN CubeCell (ASR(6501)):

equipment	Tension (V)	Current – transmission (mA)	Current reception (mA)	Current - deep_sleep
RN2903 (Microchip)	3,3	124,4	13,5	2 mA
CubeCell - ASR6501]	3,3	110,0	15,0	10 µA

Battery consumption depends mainly on several factors: the amount of invisible data (in **number of messages** and/or **message size**), as well as the **transmission power** required to transmit this data, and the **spread spectrum factor** (**SF**)
In order to optimize the consumption, the protocol LoRaWan allows the adaptation of the SF and to reduce it in order to save the peripheral power, as well as freeing up radio bandwidth and therefore limiting collisions.

2.1.5 Geo-localization

One of the specificities of the LoRaWan network is the possibility of geolocating objects using a **TDOA** (**Time Difference Of Arrival**) type technique. The different Gateways that receive the same messages from an object very precisely **timestamp** the time of receipt of this message. The distance between the object and the antenna being proportional to the time it takes for the message to be received by the antenna, solving an equation with several unknowns makes it **possible to deduce the position of the object** provided that the messages from it are received by **at least three different antennas**.

2.1.6 Capacity

The maximum occupation time of the radio channel (**duty-cycle**) imposed by the use of free frequencies is a factor limiting the number of packets that can be transmitted by a device. For example, the result of limiting to **1% on the 868 band** is a transmission time of **36 seconds per hour per channel for each terminal**.
The use of this transmission time is variable according to the spreading factor chosen. Indeed, the greater the spreading factor, the longer the time to transmit a packet will be. Also, high spreading factors are used more often than short factors in a typical network. The table below provides several examples:

Size of data	Spreading Factor	Transmission time
20 octets	10	0,4 sec
20 octets	12	1,5 sec
40 octets	10	0,5 sec
40 octets	12	1,8 sec

A study conducted in 2017 by Ferran Adelantado shows the evolution of the rate of successfully received packets according to the number of devices connected to a gateway using 3 channels. Logically, the number of packets received decreases due to collisions because the probability that several devices use the same SF simultaneously on the same channel increases.

Generally speaking, the loss of data inherent in the use of protocol on free frequencies can be solved in two ways:

- with an **acknowledgment of the data, and a repetition** if the packet has not been received, and therefore not acknowledged. This guarantee of reception not only entails a higher cost due to the use of the "**downlink**" message as well as a potentially greater occupation of the radio spectrum.
- by **data redundancy** during subsequent messages. For example, a sensor can provide the current data that it measures, as well as the previous data in order to allow continuity in the data

The choice of one or the other of the methods depends on the use cases.
Another way to increase the capacity of a LoRaWan network is to **increase the density of the antennas**, thus making it possible to **reduce the spreading factor** of the sensors and therefore **to free up bandwidth**.

2.1.7 Security protocols

LoRaWAN implements **several keys**, specific to each terminal equipment, in order to ensure the security of exchanges at the network and application level.

A **128 bit long AES ke**y called **AppKey** is used to generate the **NwkSKey** and **AppSKey**.

The **NwkSKey** key is used by the server and the end equipment to generate the **MIC integrity field** present in the packets. This field ensures that the packet has not been modified during transfer by malicious equipment. This key is also used to encrypt the content of messages containing only MAC commands.

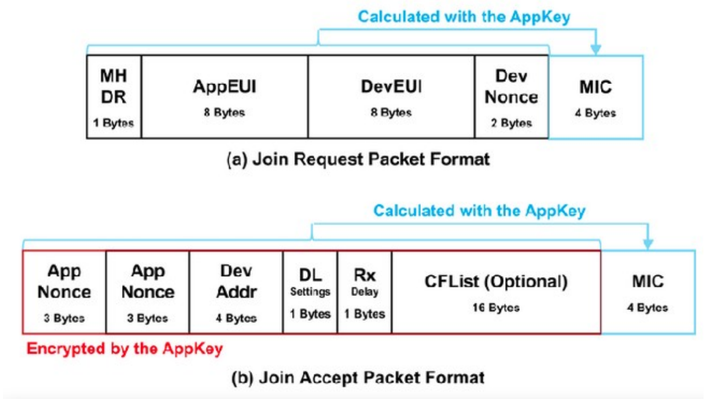
The **AppSKey** key is used to **encrypt the application data** present in the package. This key only ensures the **confidentiality** of the content of the message but **not its integrity**, which means that if the network and application servers are distinct, the network server is able to modify the content of the message.
Therefore, the LoRaWAN specification recommends using additional end-to-end protection methods for applications that would require a higher degree of security.

In order to uniquely ensure the **identity of endpoints and applications**, the protocol also defines the **`devEUI`** and **`appEUI`** fields each with a length of **64 bits**.

- **`devEUI`** makes it possible to **identify the equipment** and
- **`appEUI`** makes it possible to **identify the application** which will process its network access request.

These fields are configured in the devices.

Note that during the **OTAA activation procedure**, the equipment sends an **unencrypted** request to the server containing the **`devEUI`** and **`appEUI`** fields as well as a 16-bit random number.



The server checks whether the equipment has previously used the **random number** before accepting its request. If the number was previously used by the device, the server can implement 2 behaviors:

- It does not process the request and processes the next request if it has a valid number;
- It does not process the request and permanently excludes the equipment from the network.

2.2 LoRaWAN Gateway with RAK2245 Pi HAT and Raspberry Pi 4

Here it is the RAK Wireless RAK2245 Pi Hat Edition:

The RAK2245 is a Raspberry Pi HAT featuring a LoRaWAN multichannel concentrator module (**SX1301**) and a GPS module (Ublox MAX-7Q).

There multiple version with support for all the major frequency regions (EU433, CN470, IN865, **EU868**, AU915, US915, KR920, AS920 and AS923).

The RAK2245 comes with the following accessories

- **LoRA** antenna
- **GPS** antenna

2.2.1 Setup the Hardware

Step 1 – Insert the micro SD card with Raspbian OS (32-bit) on to your Raspberry Pi. Connect your HDMI cable and USB mouse and Keyboard, if you have Ethernet then you can also connect this now.
Once you have connected all your peripherals to the Raspberry Pi, go ahead and connect the power supply, which should boot up the Pi.

Step 2 – Before we connect the RAK831 concentrator board to the Raspberry Pi you should connect the antenna that came supplied in your kit. Make sure that the antenna is screwed in to the connector all the way but don't over tighten it.

Step 3 – We need to enable the SPI serial on the Raspberry Pi so we can communicate with the RAK831 module.
To enable SPI do the following:

- Run **sudo raspi-config** from the terminal window
- Use the down arrow to select **9 Advanced Options**
- Arrow down to **A6 SPI**
- Select **Yes** wen it asks you to enable SPI
- Also select **Yes** when it asks about automatically loading the kernel module
- Use the right arrow to navigate to **<Finish>** button
- Select **Yes** to reboot

The system will then reboot and when it comes back on, enter the following command in the terminal window:
```
ls /dev/*spi*
```

The Raspberry Pi should respond with the following:
```
/dev/spidev0.0 /dev/spidev0.1
```

These represent the SPI devices on chip enable pins 0 and 1, which are hardwired into the Raspberry Pi.

2.2.2 Software Installation – RAK Gateway (packet forwarder)

Step 1 – Make sure that your Raspberry Pi is up to date with the latest software by running the following command, whilst making sure you have git installed:

```
sudo apt-get update
sudo apt-get install git
```

Step 2 – Clone the RAK831 gateway software to your Raspberry Pi and start the installer with the following command:

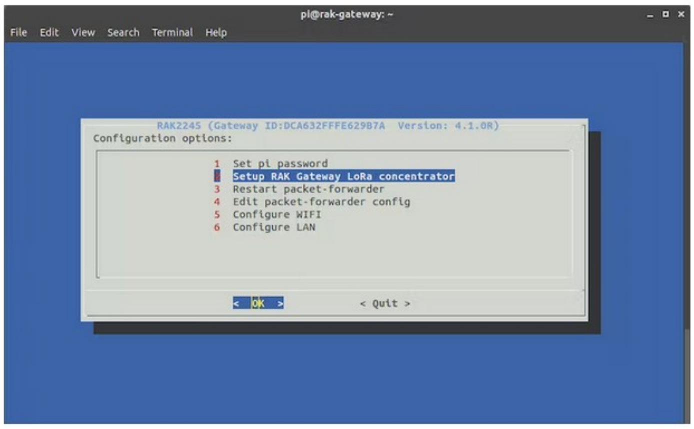
```
  $ sudo apt update; sudo apt install git -y
  $ git clone https://github.com/RAKWireless/rak_common_for_gateway.git ~/rak_common_for_gateway
  $ cd ~/rak_common_for_gateway
  $ sudo ./install.sh

Please select your gateway model:
  *      1.RAK2245
  *      2.RAK7243/RAK7244 no LTE
  *      3.RAK7243/RAK7244 with LTE
  *      4.RAK2247(USB)
  *      5.RAK2247(SPI)
  *      6.RAK2246
  *      7.RAK7248 no LTE (RAK2287 SPI + raspberry pi)
  *      8.RAK7248 with LTE (RAK2287 SPI + LTE + raspberry pi)
  *      9.RAK2287 USB
  *      10.RAK5146 USB
  *      11.RAK5146 SPI
  *      12.RAK5146 SPI with LTE
  Please enter 1-12 to select the model:
```

```
Choose: 1 (6)ip
```

The installer will present you with a **EUI number**, which at this point you should make a note of as you will need it to register your gateway with **TTN**.

When prompted to use the remote setting file enter **N** at this point.

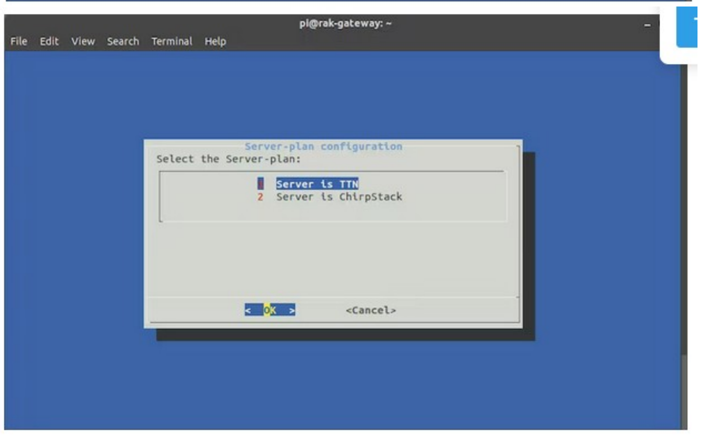
Step 5 : Wait a moment and the installation is complete.
Step 6 : reboot your gateway.
Step 7 : Now you can use "`sudo gateway-config`" to configure your gateway.



```
RAK2245 (Gateway ID:DCA632FFFE629B7A  Version: 4.1.0R)
Configuration options:
    1  Set pi password
    2  Setup RAK Gateway LoRa concentrator
    3  Restart packet-forwarder
    4  Edit packet-forwarder config
    5  Configure WIFI
    6  Configure LAN

        <  OK  >          < Quit >
```

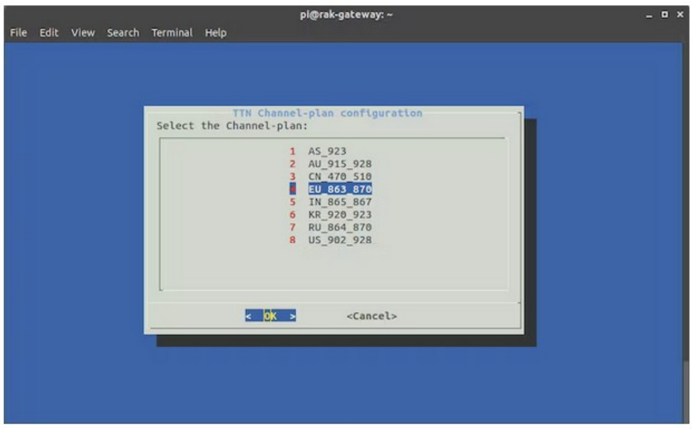
First, we need to configure the RAK Gateway LoRa concentrator (packet-forwarder):

- set the Server-plan to **TTN**



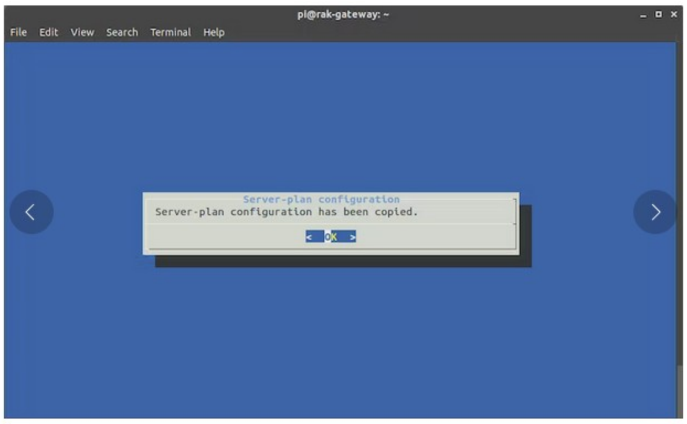
```
Server-plan configuration
Select the Server-plan:
    1  Server is TTN
    2  Server is ChirpStack

        <  OK  >          <Cancel>
```

select your region and frequency range:



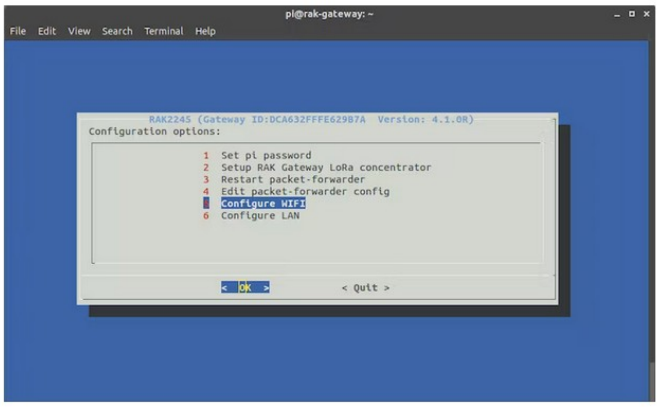
```
TTN Channel-plan configuration
Select the Channel-plan:

        1  AS_923
        2  AU_915_928
        3  CN_470_510
        4  EU_863_870
        5  IN_865_867
        6  KR_920_923
        7  RU_864_870
        8  US_902_928

     <  OK  >          <Cancel>
```

apply the configuration and restart the packet forwarder:



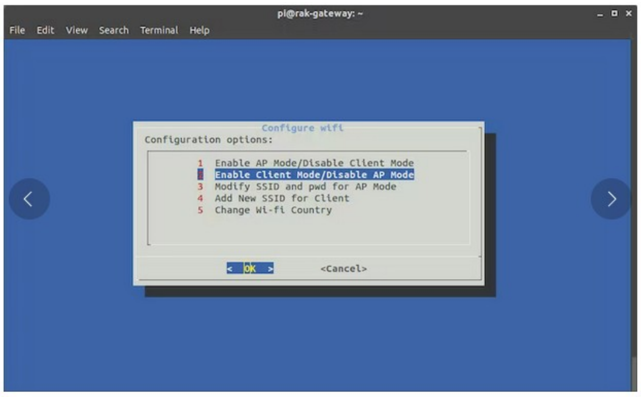
```
Server-plan configuration
Server-plan configuration has been copied.
                  <  OK  >
```

Finally, we need to configure the WiFi, or wired LAN if you prefer:



```
RAK2245 (Gateway ID:DCA632FFFE629B7A  Version: 4.1.0R)
Configuration options:

            1  Set pi password
            2  Setup RAK Gateway LoRa concentrator
            3  Restart packet-forwarder
            4  Edit packet-forwarder config
            5  Configure WIFI
            6  Configure LAN

     <  OK  >          < Quit >
```

Switch WiFi from Access Point mode to Client mode



```
Configure wifi
Configuration options:

        1  Enable AP Mode/Disable Client Mode
        2  Enable Client Mode/Disable AP Mode
        3  Modify SSID and pwd for AP Mode
        4  Add New SSID for Client
        5  Change Wi-fi Country

     <  OK  >          <Cancel>
```

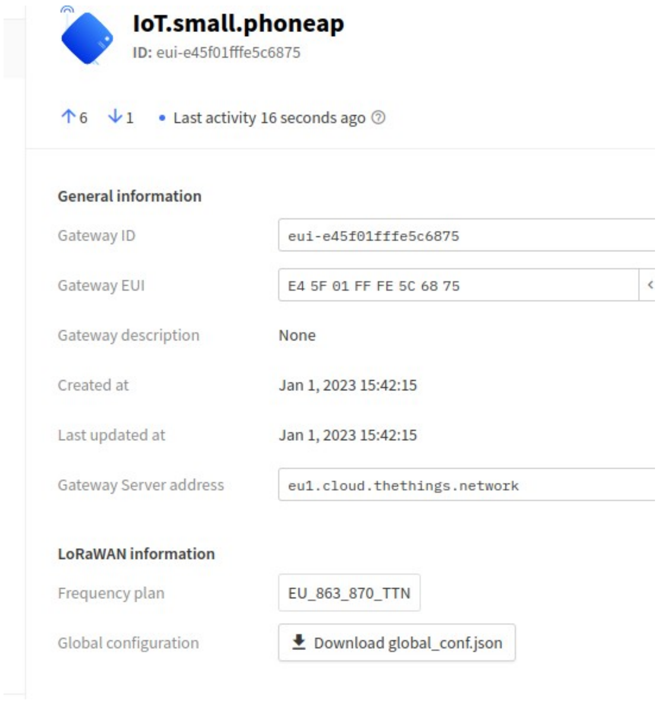
Now we are ready to prepare our devices.

2.2.2.1 Configuring and launching the packet forwarder

The packet forwarder is installed here:

```
/opt/ttn-gateway/packet_forwarder/lora_pkt_fwd/start.sh
```

After the registration of the Gateway on TTN server with the EUI identifier created from MAC address of the network interface (**Wlan0**/**Eth0**) with added **`FFFE`** sequence in the middle of the MAC address (in hexadecimal) (for example: **`E45F01FFFE5C6875)`** we can download **`global_conf.json`** file



IoT.small.phoneap
ID: eui-e45f01fffe5c6875

↑6 ↓1 • Last activity 16 seconds ago

General information

Gateway ID	eui-e45f01fffe5c6875
Gateway EUI	E4 5F 01 FF FE 5C 68 75
Gateway description	None
Created at	Jan 1, 2023 15:42:15
Last updated at	Jan 1, 2023 15:42:15
Gateway Server address	eu1.cloud.thethings.network

LoRaWAN information

Frequency plan	EU_863_870_TTN
Global configuration	Download global_conf.json

This file should replace the existing **`global_conf.json`** file in the:

`/opt/ttn-gateway/packet_forwarder/lora_pkt_fwd/` directory.

The last section of this file contains:

```
….
  "gateway_conf": {
    "gateway_ID": "E45F01FFFE5C6875",
    "server_address": "eu1.cloud.thethings.network",
    "serv_port_up": 1700,
    "serv_port_down": 1700,
    "servers": [
      {
        "gateway_ID": "E45F01FFFE5C6875",
        "server_address": "eu1.cloud.thethings.network",
        "serv_port_up": 1700,
        "serv_port_down": 1700,
        "serv_enabled": true
      }
    ]
  }
}
```

Now we can launch the packet forwarder:

```
bako@rak-gateway:/opt/ttn-gateway/packet_forwarder/lora_pkt_fwd $ ls
cfg  global_conf  global_conf.json  inc  local_conf.json  lora_pkt_fwd  Makefile  obj  readme.md
set_eui.sh  src  start.sh  update_gwid.sh

bako@rak-gateway:/opt/ttn-gateway/packet_forwarder/lora_pkt_fwd $

bako@rak-gateway:/opt/ttn-gateway/packet_forwarder/lora_pkt_fwd $ sudo ./lora_pkt_fwd
```

After the initialization of the packet forwarder we may observe some printouts:

```
beaconing datarate is set to SF9
INFO: Beaconing modulation bandwidth is set to 125000Hz
INFO: Beaconing TX power is set to 27dBm
INFO: Auto-quit after 20 non-acknowledged PULL_DATA
INFO: found local configuration file local_conf.json, parsing it
INFO: redefined parameters will overwrite global parameters
INFO: local_conf.json does not contain a JSON object named SX1301_conf
INFO: local_conf.json does contain a JSON object named gateway_conf, parsing gateway parameters
INFO: gateway MAC address is configured to E45F01FFFE5C6875
INFO: packets received with a valid CRC will be forwarded
INFO: packets received with a CRC error will NOT be forwarded
INFO: packets received with no CRC will NOT be forwarded
INFO: [main] TTY port /dev/ttyAMA0 open for GPS synchronization
INFO: [main] concentrator started, packet can now be received

INFO: Disabling GPS mode for concentrator's counter...
INFO: host/sx1301 time offset=(1672585439s:382535µs) - drift=620268551µs
INFO: Enabling GPS mode for concentrator's counter.

INFO: [down] PULL_ACK received in 110 ms



p: {"stat":{"time":"2023-01-01 15:05:31
GMT","rxnb":0,"rxok":0,"rxfw":0,"ackr":0.0,"dwnb":0,"txnb":0}}
INFO: [down] PULL_ACK received in 95 ms
INFO: [down] PULL_ACK received in 98 ms

INFO: Received pkt from mote: 260BD92C (fcnt=9)

JSON up: {"rxpk":
[{"tmst":108100316,"chan":7,"rfch":0,"freq":867.900000,"stat":1,"modu":"LORA","datr":"SF12BW125","co
dr":"4/5","lsnr":9.5,"rssi":-57,"size":21,"data":"gCzZCyaACQABFkU3xDYN2LQSLlqZ"}]}
INFO: [down] PULL_ACK received in 128 ms

INFO: Received pkt from mote: 260BD92C (fcnt=9)

JSON up: {"rxpk":
[{"tmst":115413100,"chan":1,"rfch":1,"freq":868.300000,"stat":1,"modu":"LORA","datr":"SF12BW125","co
dr":"4/5","lsnr":11.0,"rssi":-56,"size":21,"data":"gCzZCyaACQABFkU3xDYN2LQSLlqZ"}]}

INFO: Received pkt from mote: 260B5671 (fcnt=19192)

JSON up: {"rxpk":
[{"tmst":120127163,"chan":0,"rfch":1,"freq":868.100000,"stat":1,"modu":"LORA","datr":"SF7BW125","cod
r":"4/5","lsnr":9.5,"rssi":-78,"size":21,"data":"gHFWCyaA+EoBsPijqLvEXNBWyOQG"}]}

INFO: Disabling GPS mode for concentrator's counter...
INFO: host/sx1301 time offset=(1672585439s:381903µs) - drift=-316µs
INFO: Enabling GPS mode for concentrator's counter.
```

We may distinguish the reception from two devices – `mote` identified by two addresses:

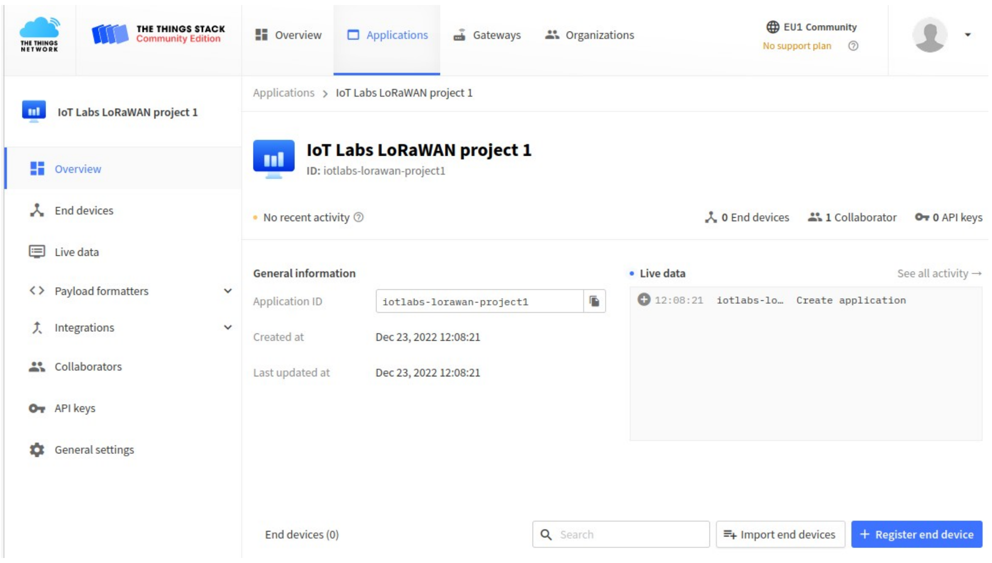
`260BD92C` **and** `260B5671`

2.2.3 TTN devices : Heltec CubeCell and ESP32 WiFi-LoRa V2

Our devices are based on the already known CubeCell and ES32 WiFi-LoRa V2 boards (see Lab3). To incorporate these devices into the TTN stack we have to follow the steps as below:

2.2.3.1 Step 1: Create an application

Once you have your TTS account registered and logged in, the next step is to create an **application**.
The application will be **home to the node devices** and allow us to capture data from them and send them to our chosen endpoint (later on in the guide).
First find the **Applications tab** within your account, and follow the user interface to add one.



IoT Labs LoRaWAN project 1
ID: iotlabs-lorawan-project1

No recent activity | 0 End devices | 1 Collaborator | 0 API keys

General information

Application ID	iotlabs-lorawan-project1
Created at	Dec 23, 2022 12:08:21
Last updated at	Dec 23, 2022 12:08:21

Live data — See all activity →

```
12:08:21  iotlabs-lo…  Create application
```

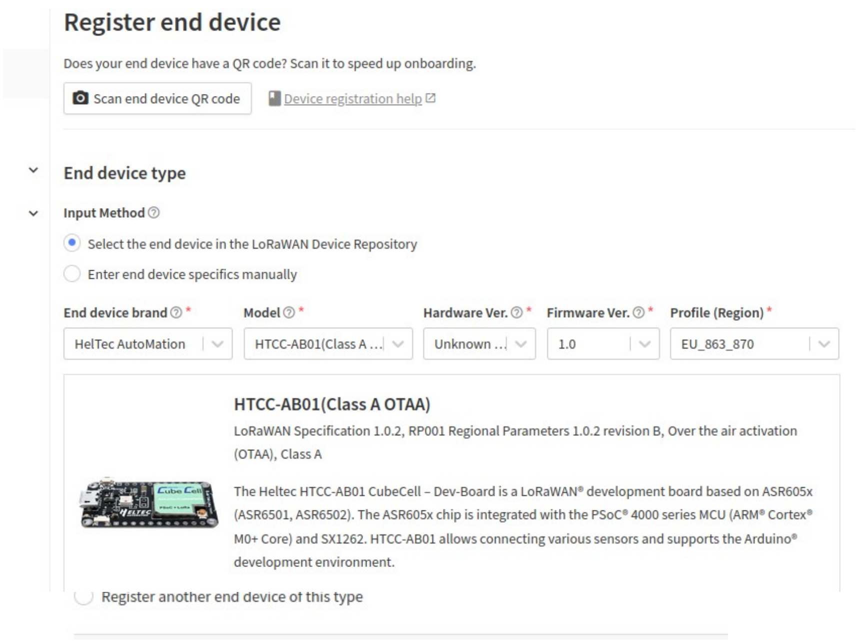
End devices (0)

Fill the form, giving an application ID and description of your choosing. Select the handler closest to your physical location from the dropdown list, and click 'Create application.'

2.2.3.2 Step 2: Register a device

Next, find devices within your newly created application and select '**Add end device**'. This will allow us to register a device and generate an identifier and key which we'll then use later when we flash the node.

For the **brand**, select **`Heltec Automation`** and the model (in case you are using the same device), **`HTTC-AB01 (Class A OTAA)`**. The frequency to select depends on where you are based.



Register end device

Does your end device have a QR code? Scan it to speed up onboarding.

End device type

Input Method

- [✓] Select the end device in the LoRaWAN Device Repository
- Enter end device specifics manually

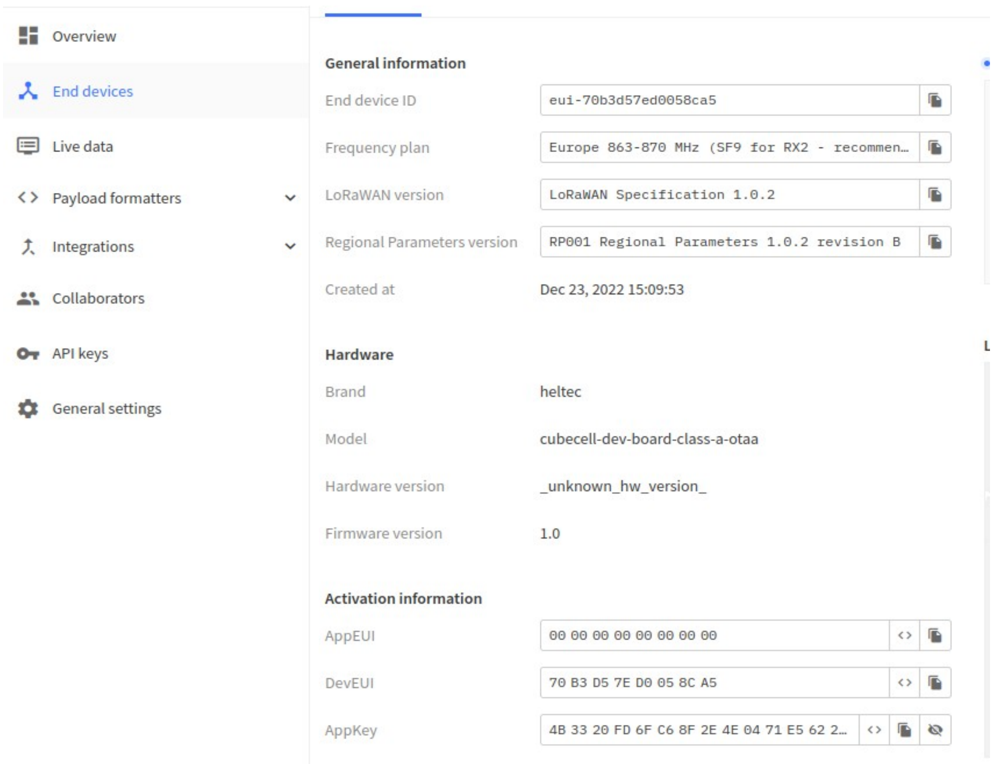
End device brand *	Model *	Hardware Ver. *	Firmware Ver. *	Profile (Region) *
HelTec AutoMation	HTCC-AB01(Class A ...	Unknown ...	1.0	EU_863_870

HTCC-AB01(Class A OTAA)
LoRaWAN Specification 1.0.2, RP001 Regional Parameters 1.0.2 revision B, Over the air activation (OTAA), Class A

The Heltec HTCC-AB01 CubeCell – Dev-Board is a LoRaWAN® development board based on ASR605x (ASR6501, ASR6502). The ASR605x chip is integrated with the PSoC® 4000 series MCU (ARM® Cortex® M0+ Core) and SX1262. HTCC-AB01 allows connecting various sensors and supports the Arduino® development environment.

Register another end device of this type

After the registration:



General information

End device ID	eui-70b3d57ed0058ca5
Frequency plan	Europe 863-870 MHz (SF9 for RX2 - recommen…
LoRaWAN version	LoRaWAN Specification 1.0.2
Regional Parameters version	RP001 Regional Parameters 1.0.2 revision B
Created at	Dec 23, 2022 15:09:53

Hardware

Brand	heltec
Model	cubecell-dev-board-class-a-otaa
Hardware version	_unknown_hw_version_
Firmware version	1.0

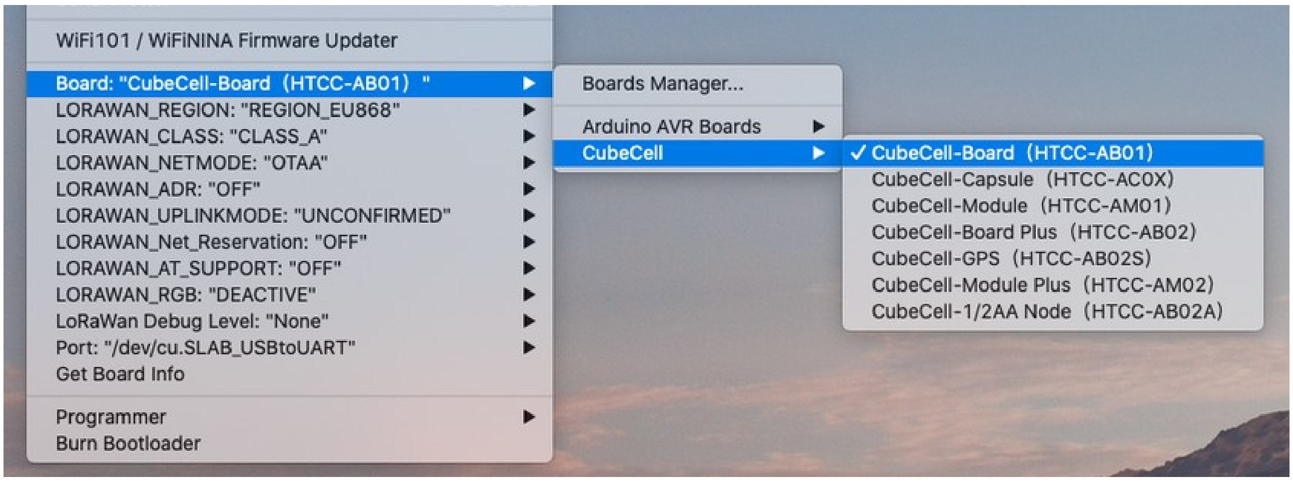
Activation information

AppEUI	00 00 00 00 00 00 00 00
DevEUI	70 B3 D5 7E D0 05 8C A5
AppKey	4B 33 20 FD 6F C6 8F 2E 4E 04 71 E5 62 2…

Activation information provides you with the parameters to be used in the code run on the registered device:

```
/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8C, 0xA5 };
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x4B, 0x33, 0x20, 0xFD, 0x6F, 0xC6, 0x8F, 0x2E, 0x4E, 0x04, 0x71, 0xE5, 0x62,
0x2C, 0xAA, 0x35 };
```

Note that the above parameters may be stored (**for this device**) in a separate file, such as `ttnparams.h`.



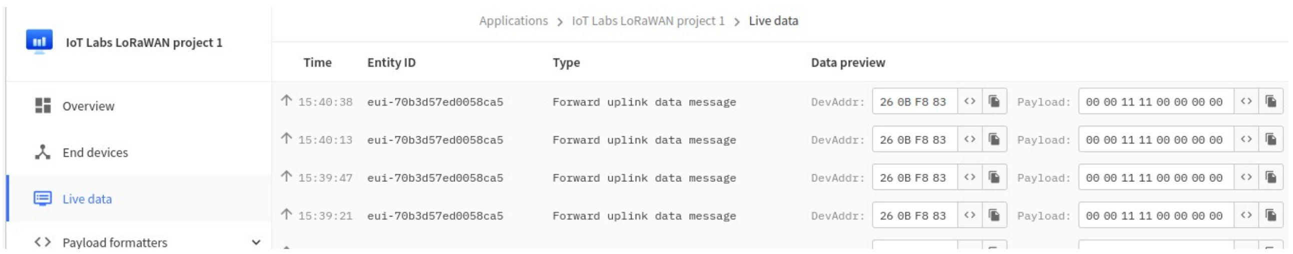
WiFi101 / WiFiNINA Firmware Updater
Board: "CubeCell-Board (HTCC-AB01) "
LORAWAN_REGION: "REGION_EU868"
LORAWAN_CLASS: "CLASS_A"
LORAWAN_NETMODE: "OTAA"
LORAWAN_ADR: "OFF"
LORAWAN_UPLINKMODE: "UNCONFIRMED"
LORAWAN_Net_Reservation: "OFF"
LORAWAN_AT_SUPPORT: "OFF"
LORAWAN_RGB: "DEACTIVE"
LoRaWan Debug Level: "None"
Port: "/dev/cu.SLAB_USBtoUART"
Get Board Info
Programmer
Burn Bootloader

Boards Manager...
Arduino AVR Boards
CubeCell
✓ CubeCell-Board (HTCC-AB01)
CubeCell-Capsule (HTCC-AC0X)
CubeCell-Module (HTCC-AM01)
CubeCell-Board Plus (HTCC-AB02)
CubeCell-GPS (HTCC-AB02S)
CubeCell-Module Plus (HTCC-AM02)
CubeCell-1/2AA Node (HTCC-AB02A)

The above are parameters used to flash the board :

- **LORAWAN_UPLINKMODE** to **confirmed**
- **LORAWAN_RGB** to **activated**

After the confirmed reception , **RGB** signal is **green**.

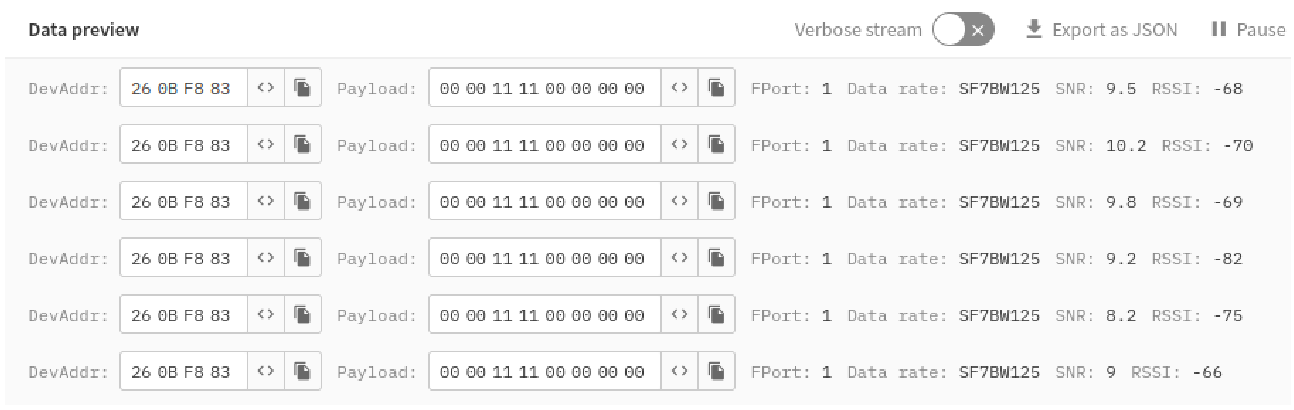

Applications > IoT Labs LoRaWAN project 1 > Live data

IoT Labs LoRaWAN project 1

Overview
End devices
Live data
Payload formatters

Time	Entity ID	Type	Data preview
15:40:38	eui-70b3d57ed0058ca5	Forward uplink data message	DevAddr: 26 0B F8 83 Payload: 00 00 11 11 00 00 00 00
15:40:13	eui-70b3d57ed0058ca5	Forward uplink data message	DevAddr: 26 0B F8 83 Payload: 00 00 11 11 00 00 00 00
15:39:47	eui-70b3d57ed0058ca5	Forward uplink data message	DevAddr: 26 0B F8 83 Payload: 00 00 11 11 00 00 00 00
15:39:21	eui-70b3d57ed0058ca5	Forward uplink data message	DevAddr: 26 0B F8 83 Payload: 00 00 11 11 00 00 00 00

This printout shows the raw data received from the device on network server (hexadecimal code)


Data preview   Verbose stream   Export as JSON   Pause

```
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 9.5  RSSI: -68
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 10.2  RSSI: -70
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 9.8  RSSI: -69
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 9.2  RSSI: -82
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 8.2  RSSI: -75
DevAddr: 26 0B F8 83  Payload: 00 00 11 11 00 00 00 00  FPort: 1  Data rate: SF7BW125  SNR: 9  RSSI: -66
```

The figures above show the **uplink** data messages received on the server in **`IoT Labs LoRaWAN project1`**. Note the payload value that corresponds to the message sent by the board:

2.2.3.3 Example code on CellCube board

The following code activates all necessary components of LoRaWAN protocol – class A, activated with **OTAA** addressing mode. The program activates the communication and sends periodically the sensor (SHT21) values (temperature/humidity) in data – payload packets. (see **`my_sht21.h`** driver presented later)

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <Wire.h>
#include "my_sht21.h"    // our SHT21 driver file
/*
 * set LoraWan_RGB to Active,the RGB active in loraWan
 * RGB red means sending;
 * RGB purple means joined done;
 * RGB blue means RxWindow1;
 * RGB yellow means RxWindow2;
 * RGB green means received done;
 */

/* OTAA para: 1st node */   // we are using this mode – see the parameters
//uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8C, 0xA5 };
//uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
//uint8_t appKey[] = { 0x4B, 0x33, 0x20, 0xFD, 0x6F, 0xC6, 0x8F, 0x2E, 0x4E, 0x04, 0x71, 0xE5, 0x62,
0x2C, 0xAA, 0x35 };

/* OTAA para: 2nd node */   // we are using this mode – see the parameters
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8C, 0xD4 };
//70B3D57ED0058CD4
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0xE6, 0xBC, 0x06, 0xA2, 0x6B, 0x8A, 0x92, 0x6B, 0x4A, 0x4C, 0x84, 0x69, 0x33,
0x70, 0x9D, 0xCE };
//E6BC06A26B8A926B4A4C846933709DCE

/* ABP para*/
uint8_t nwkSKey[] = { 0x15, 0xb1, 0xd0, 0xef, 0xa4, 0x63, 0xdf, 0xbe, 0x3d, 0x11, 0x18, 0x1e, 0x1e,
0xc7, 0xda,0x85 };
uint8_t appSKey[] = { 0xd7, 0x2c, 0x78, 0x75, 0x8c, 0xdc, 0xca, 0xbf, 0x55, 0xee, 0x4a, 0x77, 0x8d,
0x16, 0xef,0x67 };
uint32_t devAddr =  ( uint32_t )0x007e6ae1;  // 260B09BF
```

```
uint16_t userChannelsMask[6]={ 0x00FF,0x0000,0x0000,0x0000,0x0000,0x0000 };

/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;
/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = LORAWAN_CLASS;
/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 25000;
/*OTAA or ABP*/
bool overTheAirActivation = LORAWAN_NETMODE;
/*ADR enable*/
bool loraWanAdr = LORAWAN_ADR;
/* set LORAWAN_Net_Reserve ON, the node could save the network info to flash, when node reset not
need to join again */
bool keepNet = LORAWAN_NET_RESERVE;
/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = LORAWAN_UPLINKMODE;

/* Application port: Fport allows to distinguish the devices */
uint8_t appPort = 1;

uint8_t confirmedNbTrials = 8;
uint8_t i=0;

union {
  uint8_t frame[8];
  uint16_t sensor[4];
} sp;  // sent packet

/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
  uint16_t t,h;
  pinMode(Vext, OUTPUT);
  digitalWrite(Vext, LOW);
  delay(50);
  Wire.begin();  delay(50);
    Serial.println();
    t = (uint16_t) sht21_temperature();  delay(100);
    Serial.println(t);
    h = (uint16_t)sht21_humidity(); delay(100);
    Serial.println(h);
    appDataSize = 8;
    sp.sensor[0]= t;
    sp.sensor[1]= h;
    sp.sensor[2]= 0x0101; //t;
    sp.sensor[3]= 0x1010;  //h;
  Wire.end();
  digitalWrite(Vext, HIGH);
  delay(50);
  memcpy(appData,sp.frame,8);
}

void setup() {
  boardInitMcu();
  Serial.begin(115200);
  delay(400);
  Serial.println("start");
#if(AT_SUPPORT)
  enableAt();
#endif
  deviceState = DEVICE_STATE_INIT;
  LoRaWAN.ifskipjoin();
}

void loop()
{
  switch( deviceState )
  {
    case DEVICE_STATE_INIT:
    {
#if(AT_SUPPORT)
      getDevParam();
#endif
      printDevParam();
      LoRaWAN.init(loraWanClass,loraWanRegion);
      deviceState = DEVICE_STATE_JOIN;
      Serial.println("DEVICE_STATE_INIT");
      break;
    }
    case DEVICE_STATE_JOIN:
    {
      LoRaWAN.join();
      Serial.println("DEVICE_STATE_JOIN");
      break;
    }
    case DEVICE_STATE_SEND:
    {
      prepareTxFrame(appPort);
      LoRaWAN.send();
      deviceState = DEVICE_STATE_CYCLE;
     Serial.println("DEVICE_STATE_SEND");
      break;
    }
    case DEVICE_STATE_CYCLE:
    {
      // Schedule next packet transmission
      txDutyCycleTime = appTxDutyCycle + randr( 0, APP_TX_DUTYCYCLE_RND );
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      Serial.println("DEVICE_STATE_CYCLE");
      break;
    }
    case DEVICE_STATE_SLEEP:
    {
      LoRaWAN.sleep();
      //Serial.println("DEVICE_STATE_SLEEP");
      break;
    }
    default:
    {
      deviceState = DEVICE_STATE_INIT;
      Serial.println("DEFAULT");
      break;
    }
  }
}
```

Some elements of the code:

In the declaration part:

```
/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8C, 0xA5 };
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x4B, 0x33, 0x20, 0xFD, 0x6F, 0xC6, 0x8F, 0x2E, 0x4E, 0x04, 0x71, 0xE5, 0x62,
0x2C, 0xAA, 0x35 };
```

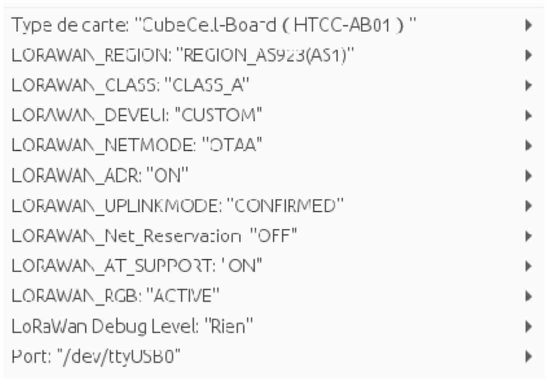
Data structure and `TxFrame` preparation:

```
union {
  uint8_t frame[8];
  uint16_t sensor[4];
  char  str[8];
} sp;  // sent packet

/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
    appDataSize = 8;
       sp.sensor[0]=(uint16_t)0x0000;  //sht.getTemperature();
       sp.sensor[1]=(uint16_t)0x1111; //sht.getHumidity();
    memcpy(appData,sp.frame,8);
}
..
```

The main loop (we use **`appPort`**):

```
    case DEVICE_STATE_SEND:
    {
      prepareTxFrame(appPort);
      LoRaWAN.send();
      deviceState = DEVICE_STATE_CYCLE;
     Serial.println("DEVICE_STATE_SEND");
      break;
    }
..
```

2.2.3.4 Compilation parameters to be chosen in Tools (Outils)



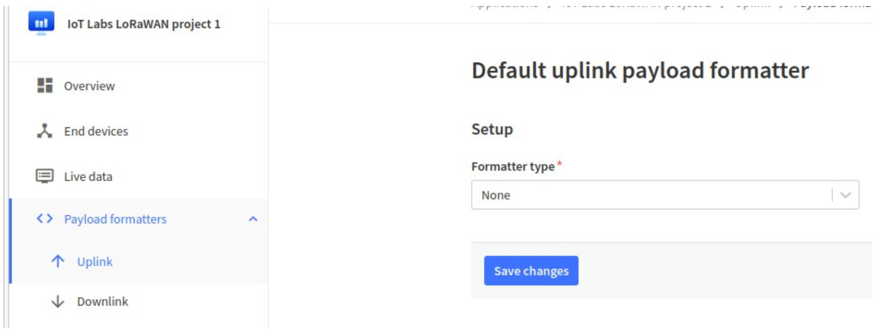
```
Type de carte: "CubeCell-Board (HTCC-AB01)"
LORAWAN_REGION: "REGION_AS923(AS1)"
LORAWAN_CLASS: "CLASS_A"
LORAWAN_DEVEUI: "CUSTOM"
LORAWAN_NETMODE: "OTAA"
LORAWAN_ADR: "ON"
LORAWAN_UPLINKMODE: "CONFIRMED"
LORAWAN_Net_Reservation: "OFF"
LORAWAN_AT_SUPPORT: "ON"
LORAWAN_RGB: "ACTIVE"
LoRaWan Debug Level: "Rien"
Port: "/dev/ttyUSB0"
```

```
/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION; // "REGION_EU868"
/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = LORAWAN_CLASS;   // "CLASS_A"
/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 25000;              // to be modified !!!
/*OTAA or ABP*/
bool overTheAirActivation = LORAWAN_NETMODE;   // "OTAA"
/*ADR enable*/
bool loraWanAdr = LORAWAN_ADR;                // "ON"
/* set LORAWAN_Net_Reserve ON, the node could save the network info to flash, when node reset not need to join again */
bool keepNet = LORAWAN_NET_RESERVE;
/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = LORAWAN_UPLINKMODE;  // "CONFIRMED"
```

2.2.3.4 Setup the payload decoder

When our device reads values from the attached sensor and sends data to TTS, it's encoded in a specific way in order to be transmitted via LoRa. When the data arrives at TTS, we can configure the application such that it knows how to decode this data and extract the sensor values.

To set this up, go to the '**Payload formatters**' section within your TTS application and add the code from the **`payload-decode.txt`** file on GitHub.

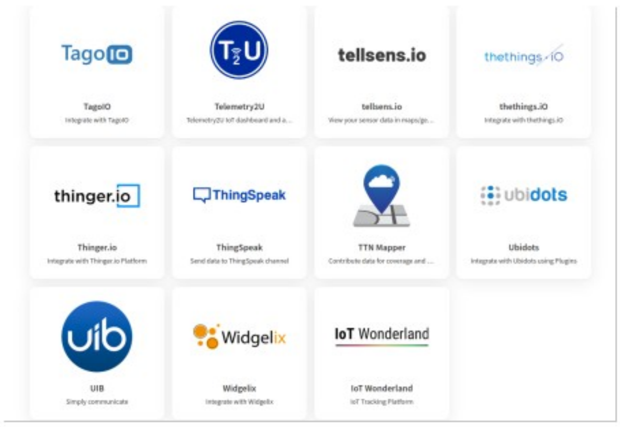


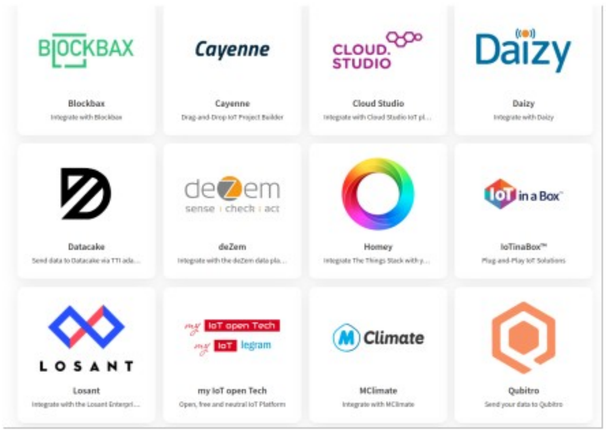
Default uplink payload formatter

Setup

Formatter type*: None

Preparing payload formatter.





The choice of Webhook applications (IoT servers). Our choice is **ThingSpeak**

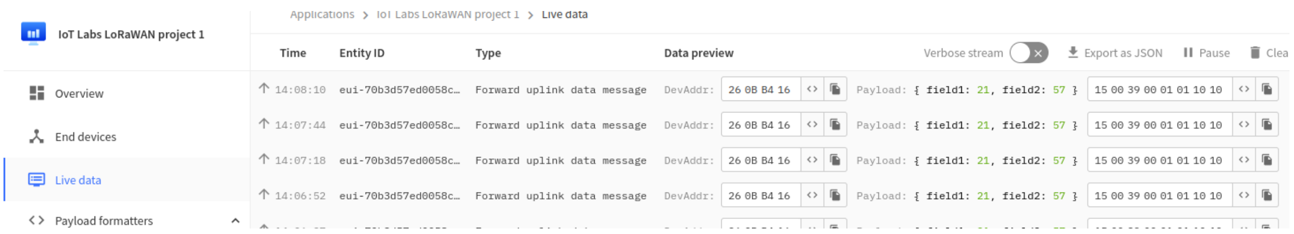
`decodeUplink()` function attached to the **Uplink** payload to the device (prepared for **ThingSpeak Webhook**) It contains the names and numbers of the **fields** within the provided **channel number** and **`Write_Key`.**
Note the use of **`input.fPort`** that indicates the formatting option for the given device.
This parameter must be defined in your device code:

```
/* Application port */
uint8_t appPort = 3;
```

The complete formatter function:

```
function decodeUplink(input)
{
    var data = {};
    var warnings = [];
    var errors = [];
    var temperature = (input.bytes[1]<<8) | input.bytes[0];
    var humidity = (input.bytes[3]<<8) | input.bytes[2];
    var fport = input.fPort;
    if(fport===3)
      {
      data.field3 = temperature;
      data.field4 = humidity;
      }
    else
      {
      data.field1 = temperature;
      data.field2 = humidity;
      }

    return {
        data: data,
        warnings: warnings,
        errors: errors
    };
}
```

Payload decoder in **JavaScript** , an example (depending on the received data - sensors):



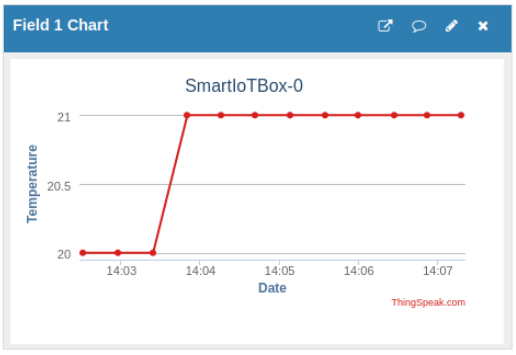
```
Time      Entity ID              Type                          Data preview
14:08:10  eui-70b3d57ed0058c…  Forward uplink data message   DevAddr: 26 0B B4 16  Payload: { field1: 21, field2: 57 }  15 00 39 00 01 01 10 10
14:07:44  eui-70b3d57ed0058c…  Forward uplink data message   DevAddr: 26 0B B4 16  Payload: { field1: 21, field2: 57 }  15 00 39 00 01 01 10 10
14:07:18  eui-70b3d57ed0058c…  Forward uplink data message   DevAddr: 26 0B B4 16  Payload: { field1: 21, field2: 57 }  15 00 39 00 01 01 10 10
14:06:52  eui-70b3d57ed0058c…  Forward uplink data message   DevAddr: 26 0B B4 16  Payload: { field1: 21, field2: 57 }  15 00 39 00 01 01 10 10
```

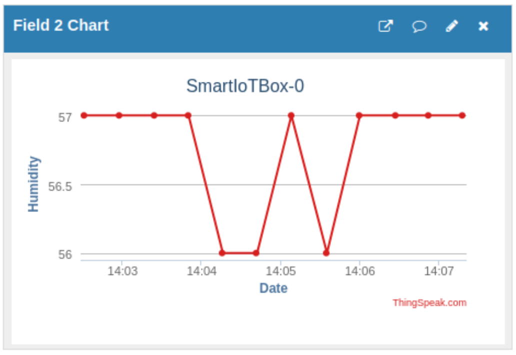
Channel Stats

Created: about a year ago

Last entry: less than a minute ago

Entries: 12





Our SHT21 driver file **`my_sht21.h`** must be stored in the same directory as **`.ino`** code

```
#define  eSHT2xAddress      0x40
#define  eTempHoldCmd       0xE3
#define  eRHumidityHoldCmd     0xE5
#define  eTempNoHoldCmd        0xF3
#define  eRHumidityNoHoldCmd   0xF5


uint16_t readSensor(uint8_t command)
{
    uint16_t result;
    Wire.beginTransmission(eSHT2xAddress);
    Wire.write(command);
    Wire.endTransmission();
    delay(100);
    Wire.requestFrom(eSHT2xAddress, 3);
    uint32_t timeout = millis() + 300;       // Don't hang here for more than 300ms
    while (Wire.available() < 3) {
        if ((millis() - timeout) > 0) {
            return 0;
        }
    }
    //Store the result
    result = Wire.read() << 8;
    result += Wire.read();
    result &= ~0x0003;    // clear two low bits (status bits)
    //Clear the final byte from the buffer
    Wire.read();
    return result;
}

float sht21_temperature(void)
{
    float value = readSensor(eTempHoldCmd);
    if (value == 0) {
        return -273;                     // Roughly Zero Kelvin indicates an error
    }
    return -46.85 + 175.72 / 65536.0 * value;
}

float sht21_humidity(void)
{
    float value = readSensor(eRHumidityHoldCmd);
    if (value == 0) {
        return 0;                        // Some unrealistic value
    }
    return -6.0 + 125.0 / 65536.0 * value;
}
```

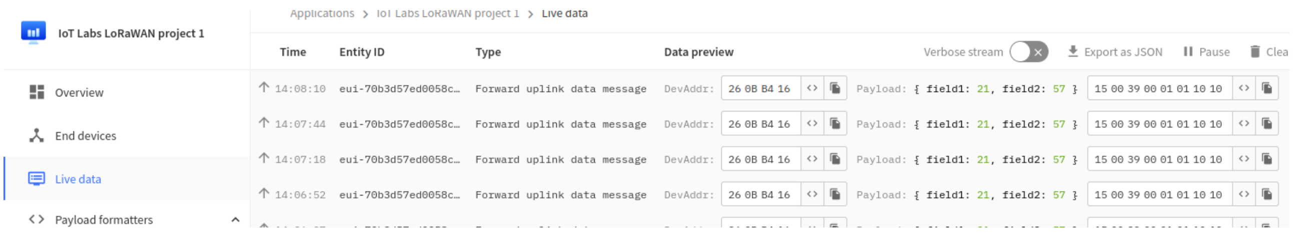
2.2.3.5 Running the application code on CubeCell board

Terminal output:

```
21
59
confirmed uplink sending ...
DEVICE_STATE_SEND
DEVICE_STAT?received unconfirmed downlink: rssi = -71, snr = 14, datarate = 5

21
58
confirmed uplink sending ...
DEVICE_STATE_SEND
DEVICE_STAT?received unconfirmed downlink: rssi = -63, snr = 13, datarate = 5
```

TTN live data output:

Payload decoder in **JavaScript** , an example (depending on the received data - sensors):

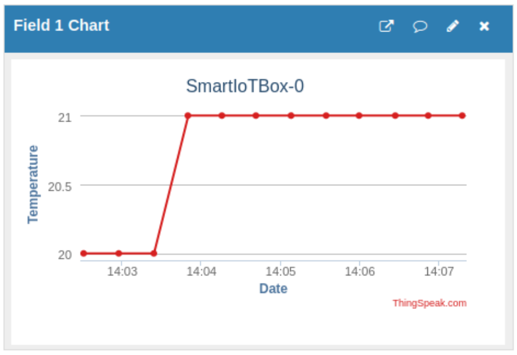


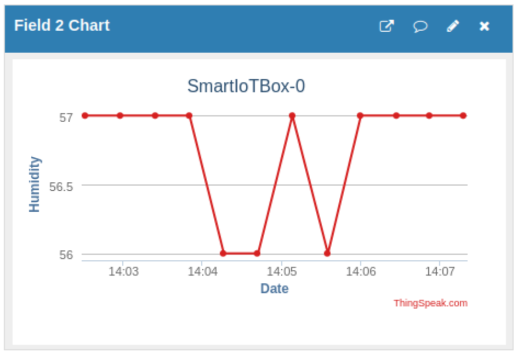
Time	Entity ID	Type	Data preview
14:08:10	eui-70b3d57ed0058c…	Forward uplink data message	DevAddr: 26 0B B4 16 Payload: { field1: 21, field2: 57 } 15 00 39 00 01 01 10 10
14:07:44	eui-70b3d57ed0058c…	Forward uplink data message	DevAddr: 26 0B B4 16 Payload: { field1: 21, field2: 57 } 15 00 39 00 01 01 10 10
14:07:18	eui-70b3d57ed0058c…	Forward uplink data message	DevAddr: 26 0B B4 16 Payload: { field1: 21, field2: 57 } 15 00 39 00 01 01 10 10
14:06:52	eui-70b3d57ed0058c…	Forward uplink data message	DevAddr: 26 0B B4 16 Payload: { field1: 21, field2: 57 } 15 00 39 00 01 01 10 10

ThingSpeak diagram:

Channel Stats

Created: about a year ago
Last entry: less than a minute ago
Entries: 12





2.2.4 Sendig LoRaWAN packets from Heltec ESP32 - LoRa (V2)

To activate the device we need to **`register`** it in the application and add it with generated **`devEUI`** and **`appKey`**. The device operates with **OTAA** addressing mode.

Modified file from:

`https://github.com/HelTecAutomation/ESP32_LoRaWAN`

2.2.4.1 Sending some data (packet)

```
#include "LoRaWan_APP.h"

/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8D, 0xBD }; // 70 B3 D5 7E D0 05 8D BD
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x1A, 0x67, 0x50, 0xEB, 0x6A, 0x6C, 0x3C, 0x93, 0x1D, 0x55, 0x82, 0xC4, 0xE8,
0xC9, 0x9E, 0xC1 };
// 1A 67 50 EB 6A 6C 3C 93 1D 55 82 C4 E8 C9 9E C1

/* ABP para*/
uint8_t nwkSKey[] = { 0x15, 0xb1, 0xd0, 0xef, 0xa4, 0x63, 0xdf, 0xbe, 0x3d, 0x11, 0x18, 0x1e, 0x1e,
0xc7, 0xda,0x85 };
uint8_t appSKey[] = { 0xd7, 0x2c, 0x78, 0x75, 0x8c, 0xdc, 0xca, 0xbf, 0x55, 0xee, 0x4a, 0x77, 0x8d,
0x16, 0xef,0x67 };
uint32_t devAddr =  ( uint32_t )0x007e6ae1;

/*LoraWan channelsmask, default channels 0-7*/
uint16_t userChannelsMask[6]={ 0x00FF,0x0000,0x0000,0x0000,0x0000,0x0000 };

/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;

/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = CLASS_A;

/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 150000;

/*OTAA or ABP*/
bool overTheAirActivation = true;

/*ADR enable*/
bool loraWanAdr = true;

/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = true;

/* Application port */
uint8_t appPort = 2;

uint8_t confirmedNbTrials = 4;

/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
    appDataSize = 4;
    appData[0] = 0x00;
    appData[1] = 0x01;
    appData[2] = 0x02;
    appData[3] = 0x03;
}

void setup() {
  Serial.begin(115200);
  SPI.begin(SCK,MISO,MOSI,SS);
  Mcu.begin();
  deviceState = DEVICE_STATE_INIT;
}
```

```
void loop()
{
  switch( deviceState )
  {
    case DEVICE_STATE_INIT:
    {
#if(LORAWAN_DEVEUI_AUTO)
      LoRaWAN.generateDeveuiByChipID();
#endif
      LoRaWAN.init(loraWanClass,loraWanRegion);
      break;
    }
    case DEVICE_STATE_JOIN:
    {
      LoRaWAN.join();
      break;
    }
    case DEVICE_STATE_SEND:
    {
      prepareTxFrame( appPort );
      LoRaWAN.send();
      deviceState = DEVICE_STATE_CYCLE;
      break;
    }
    case DEVICE_STATE_CYCLE:
    {
      // Schedule next packet transmission
      txDutyCycleTime = appTxDutyCycle + randr( -APP_TX_DUTYCYCLE_RND, APP_TX_DUTYCYCLE_RND );
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      break;
    }
    case DEVICE_STATE_SLEEP:
    {
      LoRaWAN.sleep(loraWanClass);
      break;
    }
    default:
    {
      deviceState = DEVICE_STATE_INIT;
      break;
    }
  }
}
```

2.2.4.2 Sending some data and displaying

The same code with OLED display. Note that we use predefined display functions:

`LoRaWAN.displayMcuInit();` in the `setup()`
`LoRaWAN.displayJoining();` `LoRaWAN.displaySending();` `LoRaWAN.displayAck();` in the `loop()`

```
#include "LoRaWan_APP.h"

/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8D, 0xBD }; // 70 B3 D5 7E D0 05 8D BD
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x1A, 0x67, 0x50, 0xEB, 0x6A, 0x6C, 0x3C, 0x93, 0x1D, 0x55, 0x82, 0xC4, 0xE8,
0xC9, 0x9E, 0xC1 };
// 1A 67 50 EB 6A 6C 3C 93 1D 55 82 C4 E8 C9 9E C1

/* ABP para*/
uint8_t nwkSKey[] = { 0x15, 0xb1, 0xd0, 0xef, 0xa4, 0x63, 0xdf, 0xbe, 0x3d, 0x11, 0x18, 0x1e, 0x1e,
0xc7, 0xda,0x85 };
uint8_t appSKey[] = { 0xd7, 0x2c, 0x78, 0x75, 0x8c, 0xdc, 0xca, 0xbf, 0x55, 0xee, 0x4a, 0x77, 0x8d,
0x16, 0xef,0x67 };
uint32_t devAddr =  ( uint32_t )0x007e6ae1;

/*LoraWan channelsmask, default channels 0-7*/
uint16_t userChannelsMask[6]={ 0x00FF,0x0000,0x0000,0x0000,0x0000,0x0000 };

/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;
```

```
/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = CLASS_A;

/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 150000;

/*OTAA or ABP*/
bool overTheAirActivation = true;

/*ADR enable*/
bool loraWanAdr = true;

/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = true;

/* Application port */
uint8_t appPort = 2;

uint8_t confirmedNbTrials = 4;

/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
  /*appData size is LORAWAN_APP_DATA_MAX_SIZE which is defined in "commissioning.h".
  *appDataSize max value is LORAWAN_APP_DATA_MAX_SIZE.
  *if enabled AT, don't modify LORAWAN_APP_DATA_MAX_SIZE, it may cause system hanging or failure.
  *if disabled AT, LORAWAN_APP_DATA_MAX_SIZE can be modified, the max value is reference to lorawan
region and SF.
  *for example, if use REGION_CN470,
  *the max value for different DR can be found in MaxPayloadOfDatarateCN470 refer to DataratesCN470
and BandwidthsCN470 in "RegionCN470.h".
  */
    appDataSize = 4;
    appData[0] = 0x00;
    appData[1] = 0x01;
    appData[2] = 0x02;
    appData[3] = 0x03;
}


void setup() {
  Serial.begin(115200);
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRaWAN.displayMcuInit();

  Mcu.begin();
  deviceState = DEVICE_STATE_INIT;
}

void loop()
{
  switch( deviceState )
  {
    case DEVICE_STATE_INIT:
    {
#if(LORAWAN_DEVEUI_AUTO)
      LoRaWAN.generateDeveuiByChipID();
#endif
      LoRaWAN.init(loraWanClass,loraWanRegion);
      break;
    }
    case DEVICE_STATE_JOIN:
    {
      LoRaWAN.displayJoining();
      LoRaWAN.join();
      break;
    }
    case DEVICE_STATE_SEND:
    {
      LoRaWAN.displaySending();
      prepareTxFrame( appPort );
      LoRaWAN.send();
      deviceState = DEVICE_STATE_CYCLE;
      break;
    }
    case DEVICE_STATE_CYCLE:
```

```
    {
      // Schedule next packet transmission
      txDutyCycleTime = appTxDutyCycle + randr( -APP_TX_DUTYCYCLE_RND, APP_TX_DUTYCYCLE_RND );
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      break;
    }
    case DEVICE_STATE_SLEEP:
    {
      LoRaWAN.displayAck();
      LoRaWAN.sleep(loraWanClass);
      break;
    }
    default:
    {
      deviceState = DEVICE_STATE_INIT;
      break;
    }
  }
}
```

2.2.4.3 Sending data from SHT21 sensor (no display)

The board does not allow for using I2C bu**s** and display on local OLED screen (probable conflict !).
B**ut** you can always use a separate OLED display connected **externally** to I2C bus.

```
#include "LoRaWan_APP.h"
#include <Wire.h>
#include "my_sht21.h"

#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(/* clock=*/ 15, /* data=*/ 4, /* reset=*/ 16);

/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x05, 0x8D, 0xBD }; // 70 B3 D5 7E D0 05 8D BD
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x1A, 0x67, 0x50, 0xEB, 0x6A, 0x6C, 0x3C, 0x93, 0x1D, 0x55, 0x82, 0xC4, 0xE8,
0xC9, 0x9E, 0xC1 };
// 1A 67 50 EB 6A 6C 3C 93 1D 55 82 C4 E8 C9 9E C1

/* ABP para*/
uint8_t nwkSKey[] = { 0x15, 0xb1, 0xd0, 0xef, 0xa4, 0x63, 0xdf, 0xbe, 0x3d, 0x11, 0x18, 0x1e, 0x1e,
0xc7, 0xda,0x85 };
uint8_t appSKey[] = { 0xd7, 0x2c, 0x78, 0x75, 0x8c, 0xdc, 0xca, 0xbf, 0x55, 0xee, 0x4a, 0x77, 0x8d,
0x16, 0xef,0x67 };
uint32_t devAddr =  ( uint32_t )0x007e6ae1;

/*LoraWan channelsmask, default channels 0-7*/
uint16_t userChannelsMask[6]={ 0x00FF,0x0000,0x0000,0x0000,0x0000,0x0000 };

/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;

/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = CLASS_A;

/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 40000;

/*OTAA or ABP*/
bool overTheAirActivation = true;

/*ADR enable*/
bool loraWanAdr = true;

/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = true; //false;

/* Application port */
uint8_t appPort = 2;

uint8_t confirmedNbTrials = 4;

uint16_t t,h;
```

```
void getth()
{
    Wire.begin(21,22);
    Serial.print("Humidity(%RH): ");
    Serial.print(sht21_humidity());
    h=(uint16_t)sht21_humidity();
    delay(80);
    Serial.print("     Temperature(C): ");
    Serial.println(sht21_temperature());
    t=(uint16_t)sht21_temperature();
    Wire.end();
}

void dispsmall(const char* dtape)
{
  int dlen=0,i=0;
  char ligne[24];
  u8x8.clear();
  dlen=strlen(dtape);
  for(i=0;i<8;i++)
    {
       if(dlen>(i*16))
         {
          memset(ligne,0x00,24);
          strncpy(ligne,dtape+(i*16),16);
          u8x8.drawString(0,i,ligne); delay(40);
         }
    }
}

 getth();
    appDataSize = 4;
    appData[0] = 0x00;
    appData[1] = 0x01;
    appData[2] = 0x02;
    appData[3] = 0x03;
}

void setup() {
  Serial.begin(115200);
//  u8x8.begin();  // init OLED display
//  u8x8.setFont(u8x8_font_chroma48medium8_r);
  SPI.begin(SCK,MISO,MOSI,SS);
  //LoRaWAN.displayMcuInit();
  Mcu.begin();
  deviceState = DEVICE_STATE_INIT;
  //dispsmall("LoRaWAN start");
}

void loop()
{
  switch( deviceState )
  {
    case DEVICE_STATE_INIT:
    {
#if(LORAWAN_DEVEUI_AUTO)
      LoRaWAN.generateDeveuiByChipID();
#endif
      LoRaWAN.init(loraWanClass,loraWanRegion);
      break;
    }
    case DEVICE_STATE_JOIN:
    {
      //LoRaWAN.displayJoining();
      LoRaWAN.join();
      break;
    }
    case DEVICE_STATE_SEND:
    {
      //LoRaWAN.displaySending();
      prepareTxFrame(appPort);
      LoRaWAN.send();
      deviceState = DEVICE_STATE_CYCLE;
      break;
    }
    case DEVICE_STATE_CYCLE:
    {
      // Schedule next packet transmission
      txDutyCycleTime = appTxDutyCycle + randr( -APP_TX_DUTYCYCLE_RND, APP_TX_DUTYCYCLE_RND );
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      break;
    }
    case DEVICE_STATE_SLEEP:
    {
      //LoRaWAN.displayAck();
      LoRaWAN.sleep(loraWanClass);
      break;
    }
    default:
    {
      deviceState = DEVICE_STATE_INIT;
      break;
    }
  }
}
```

To do

Prepare TTN (TTS) free account to create your application and associated devices. Prepare ThingSpeak account with channel number and write key.

1. Take a CubeCell IoT DevKit and implement the presented above code register the device in **OTAA** mode and note the generated parameters 'example:
```
       uint8_t devEui[] = { 0x70,0xB3,0xD5,0x7E,0xD0,0x05,0x8D,0xBD };
       uint8_t appEui[] = { 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 };
       uint8_t appKey[] =
{ 0x1A,0x67,0x50,0xEB,0x6A,0x6C,0x3C,0x93,0x1D,0x55,0x82,0xC4,0xE8, ..}
```
   in the first example send the measurements from an SHT21 sensor
   in the second example add the battery state
   Analyze the `Live data` for your application received on TTN server

2. Use PPK2 meter to analyze the power consumption of the application.

Table of Contents