Low Power IoT Architectures Labs

with CubeCell – ARS601 (Psoc+LoRa)

The content:

0. Introduction

The following labs are 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 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.

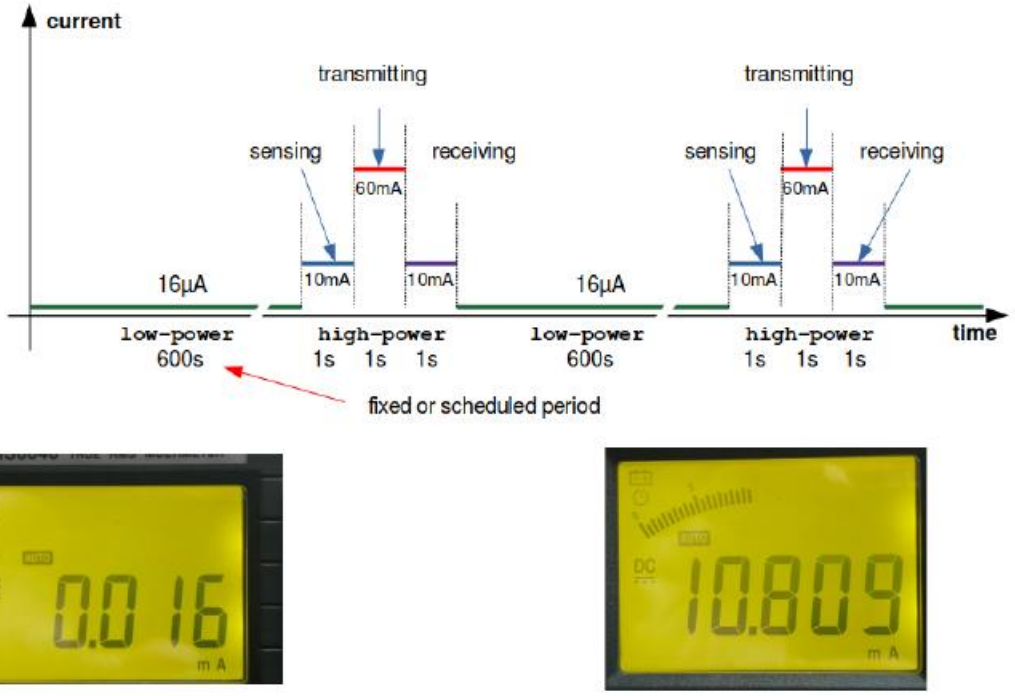


Figure 0.1 Low_power and **high_power** periods of operational cycle

0.1 Hardware Architecture - main board (CubeCell)

Heltec CubeCell board (HTCC-AB01) uses the ASR6501 module. The ASR6501 is 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.

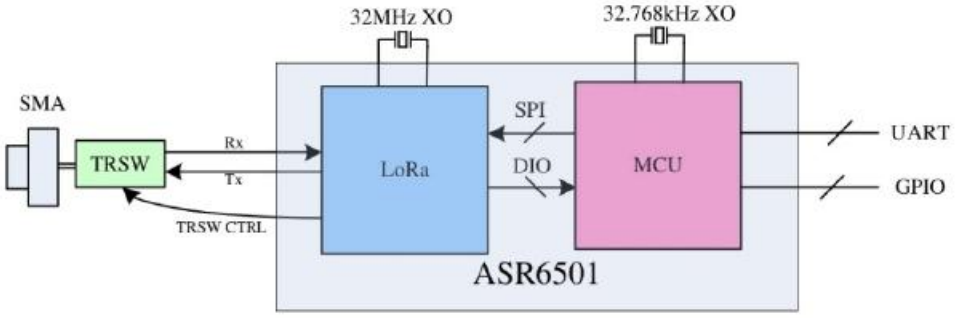


Figure 0.2 ASR6501 architectural scheme with MCU and Lora modem connected via SPI bus

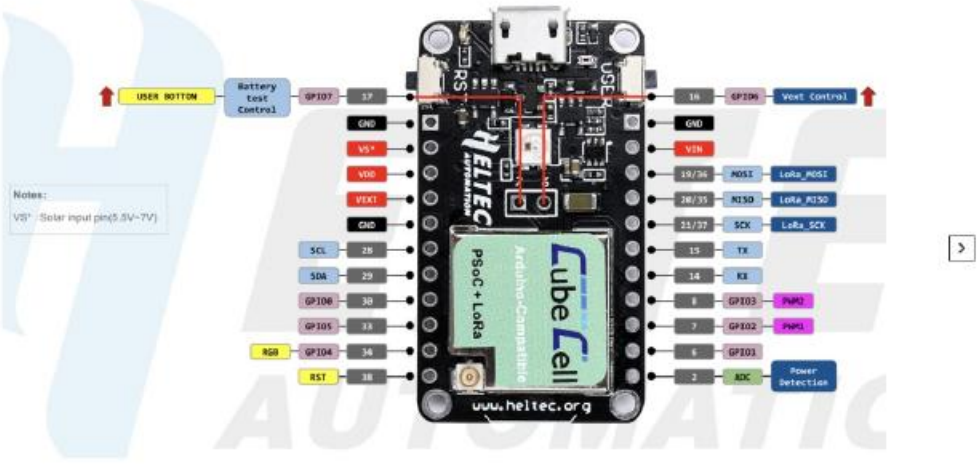


Figure 0.3 CellCube board and its pinout

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

Software installation for Debian / Ubuntu OS

- Install latest Arduino IDE from arduino.cc
- Open Terminal and execute the following command (copy->paste and hit enter):

```
sudo usermod -a -G dialout $USER && \
sudo apt-get install git && \
mkdir -p ~/Arduino/hardware/CubeCell && \
cd ~/Arduino/hardware/CubeCell && \
git clone https://github.com/HelTecAutomation/ASR650x-Arduino.git CubeCell
&& \
cd CubeCell/tools && \
python get.py
```

- Restart Arduino IDE

Software installation for Mac OS

- Install latest Arduino IDE from arduino.cc
- Open Terminal and execute the following command (copy->paste and hit enter one by one):

```
mkdir -p ~/Documents/Arduino/hardware/CubeCell
cd ~/Documents/Arduino/hardware/CubeCell
git clone https://github.com/HelTecAutomation/ASR650x-Arduino.git CubeCell
cd CubeCell/tools
python get.py
```

- Restart Arduino IDE

Principal link for software resources:

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

Lab 1 – 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.

1.1 Board chip identifier

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

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

1.2 Integrated RGB LED

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

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

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

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

1.3 Integrated USR button

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

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

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

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

void loop() {}
```

1.4 User flash memory.

The chip has 1K user flash:

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

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

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

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

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

}
```

The terminal printout:

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

1.5 Sleep modes: timer and user interruption

1.5.1 Timer mode

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

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

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

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

1.5.2 User interrupt mode

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

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

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

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

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

1.6 System time

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

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

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

1.7 WatchDog timer

Watchdog timers are commonly found in embedded systems and 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

Lab 2 - 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.

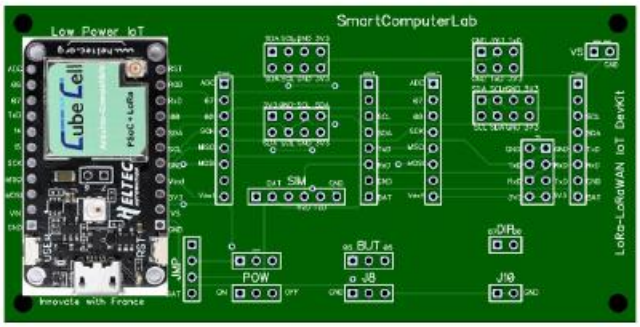
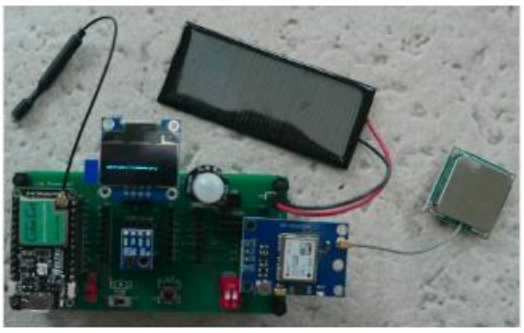


Fig 2.1 Low Power IoT DevKit from SmartComputerLab with CubeCell board

2.1 SSD1306 OLED



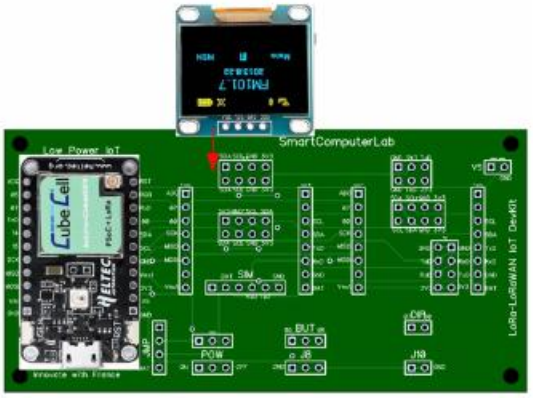


Fig 2.2 IoT DevKit with OLED screen connected via I2C bus

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

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

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

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

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

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

2.2 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 3V3
  delay(100);
  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
```

2.3 SHT21 sensor module (I2C)

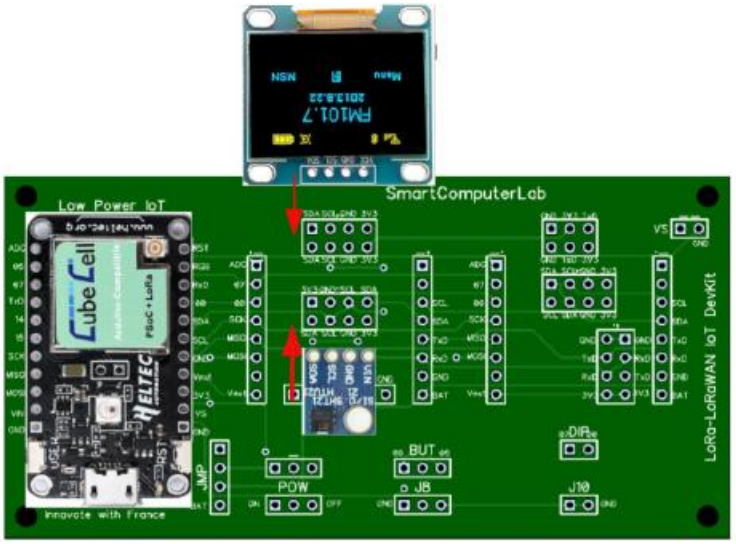


Fig 2.3 IoT DevKit with OLED screen and SHT21 sensor connected via I2C bus

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

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

SHT21 SHT21;

float t,h;

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

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

Reading SHT21 sensor and display on OLED

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

SHT21 SHT21;

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

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

```
    display.drawString(0, 15, line2);
    display.drawString(0, 30, line3);
    display.display();
}

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

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

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

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



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

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

To do

- Test the above examples
- Experiment with:
  - `pinMode(Vext, OUTPUT);`
  - `digitalWrite(Vext, LOW); delay(100);`

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

2.4 BH1750 – luminosity sensor

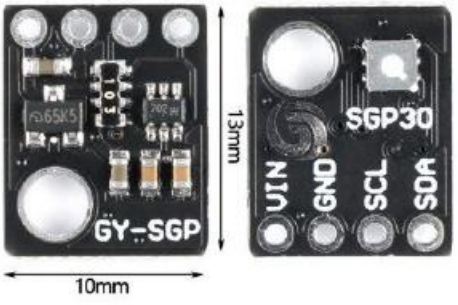

```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

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

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

To do

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

2.5 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); //
[mg/m^3]
    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

2.6 “Time of Flight” distance sensors – VL53L0X and VL53L1X

2.6.1 VL53L03

The **VL53L0X** is a *Time of Flight* distance sensor like no other you've used! The sensor 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);
}
```

2.6.2 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

2.7 `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).

2.7.1 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);
}
```

2.7.2 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);
}
```

2.7.3 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

2.8 MB85RC256V FRAM module

The first external component that will be used to store the application parameters and the data captured on the selected sensors is FRAM memory module.

The FRAM module is non-volatile and can easily be read/written 10 thousand times. The FRAM memory is similar to dynamic random access memory (DRAM), but exploits a ferro-electric layer instead of a dielectric layer. It is particularly suitable for use with low power data loggers and for buffering data in the absence of a stable voltage source.
The FRAM chip used provides 32 KB of memory and may be driven by a clock up to 20 MHz.
Each byte can be read and written instantly; it will retain memory for a long time at normal temperature.

Specification:

- Addr: 1010 + A2 + A1 + A0 // 1010000 or 0x50
- Default: 0x50
- VCC / Logic: 2.7-5.5V

2.8.1 MB85RC256V - FRAM test

```
//     URL: https://github.com/RobTillaart/FRAM_I2C
#include "FRAM.h"
FRAM fram;
uint32_t start;
uint32_t stop;

void setup()
{
  Serial.begin(9600);
  Serial.println(__FILE__);
  Serial.print("FRAM_LIB_VERSION: "); Serial.println(FRAM_LIB_VERSION);
  Wire.begin(29,28);  // default CubeCell 29 - SDA, 28 - SCL
  int rv = fram.begin(0x50);
  if (rv != 0){ Serial.println(rv); }
  else
  {
    testID();   testFRAMmemory();
    testReadWriteSmall();   testReadWriteLarge();
    testWriteText(); testReadText1(); testReadText2();
  }
  Serial.println("done...");
}

void loop(){}

void testID()
{
  Serial.println();  Serial.println(__FUNCTION__);
  Serial.println("takes ~32 seconds");
  Serial.print("ManufacturerID: "); Serial.println(fram.getManufacturerID());
  Serial.print("     ProductID: "); Serial.println(fram.getProductID());
  Serial.print("    memory size: "); Serial.println(fram.getSize());
  Serial.println();
}

void testFRAMmemory()
{
  Serial.println();
  Serial.println(__FUNCTION__);
  Serial.println("takes ~32 seconds");
  start = millis();
  uint8_t val = 0x55;
  for (uint16_t addr = 0; addr < 32768; addr++)
  {
    fram.write8(addr, val);
    if (fram.read8(addr) != 0x55)
```

```
    {
      Serial.print("FAIL: \t");  Serial.println(addr);
    }
    if (addr % 1000 == 0)  {   Serial.print("."); }
  }
  stop = millis(); Serial.println();
  Serial.print("TIME:\t");Serial.print(stop - start);
  Serial.println(" ms");Serial.println();
}

void testReadWriteSmall()
{
  Serial.println();
  Serial.println(__FUNCTION__);
  Serial.print("test8:\t");
  uint8_t t8 = 0xFE;
  fram.write8(1000, t8);
  if (fram.read8(1000) != 0xFE) { Serial.println("failed.");}
  else {  Serial.println("ok."); }
  Serial.print("test16:\t");
  uint16_t t16 = 0xFADE;
  fram.write16(1000, t16);
  if (fram.read16(1000) != 0xFADE) { Serial.println("failed.");}
  else {  Serial.println("ok.");}
  Serial.print("test32:\t");
  uint32_t t32 = 0xFADEFACE;
  fram.write32(1000, t32);
  if (fram.read32(1000) != 0xFADEFACE)  {    Serial.println("failed."); }
  else
  {  Serial.println("ok.");}
  Serial.println();
}

void testReadWriteLarge()
{
  Serial.println();
  Serial.println(__FUNCTION__);
  uint8_t ar[100];
  for (int i = 0; i < 100; i++) ar[i] = i;
  start = millis();
  fram.write(1000, ar, 100);
  stop = millis();
  Serial.print("WRITE 100 bytes TIME:\t");
  Serial.print(stop - start);
  Serial.println(" ms");
  for (int i = 0; i < 100; i++) ar[i] = 0;
  start = millis();
  fram.read(1000, ar, 100);
  stop = millis();
  Serial.print("READ 100 bytes TIME:\t");
  Serial.print(stop - start);
  Serial.println(" ms");
  for (int i = 0; i < 100; i++)
  {
    if (ar[i] != i)
    {
      Serial.print("FAIL: \t");  Serial.println(i);
    }
  }
  Serial.println();
}

void testWriteText()
{
  char str[10][20] =
  {
    "Hello world 0",
    //..
    "Hello world 9",
  };
  Serial.println();
  Serial.println(__FUNCTION__);
  start = millis();
  fram.write(2000, (uint8_t *)str, 200);
  stop = millis();
  Serial.print("WRITE 200 bytes TIME:\t");
```

```
  Serial.print(stop - start);
  Serial.println(" ms");
  Serial.println();
}

void testReadText1()
{
  char str[10][20];
  Serial.println();
  Serial.println(__FUNCTION__);
  start = millis();
  fram.read(2000, (uint8_t *)str, 200);
  stop = millis();
  Serial.print("READ 200 bytes TIME:\t");
  Serial.print(stop - start);
  Serial.println(" ms");
  Serial.println();
  for (int i = 0; i < 10; i++)
  {  Serial.println(str[i]);  }
  Serial.println();
}

void testReadText2()
{
  char str[20];
  Serial.println();
  Serial.println(__FUNCTION__);
  for (int i = 0; i < 10; i++)
  {
    fram.read(2000 + 20 * i, (uint8_t *)str, 20);
    Serial.println(str);
  }
  Serial.println();
}
```

The result displayed on IDE terminal:

```
ID: 65535
     ProductID: 65535
    memory size: 0

testFRAMmemory
takes ~32 seconds
................................
TIME:   25882 ms

testReadWriteSmall
test8: ok.
test16: ok.
test32: ok.

testReadWriteLarge
WRITE 100 bytes TIME:  8 ms
READ 100 bytes TIME:   9 ms

testWriteText
WRITE 200 bytes TIME: 16 ms

testReadText1
READ 200 bytes TIME:   18 ms
ID: 65535
     ProductID: 65535
    memory size: 0

testFRAMmemory
takes ~32 seconds
................................
TIME:   25882 ms

testReadWriteSmall
test8: ok.
test16: ok.
test32: ok.

testReadWriteLarge
```

```
WRITE 100 bytes TIME:  8 ms
READ 100 bytes TIME:   9 ms

testWriteText
WRITE 200 bytes TIME: 16 ms

testReadText1
READ 200 bytes TIME:   18 ms

Hello world 0
Hello world 1
Hello world 2
Hello world 3
ID: 65535
     ProductID: 65535
    memory size: 0


..

testReadText2
Hello world 0
..
Hello world 9

done...
Hello world 0
..
Hello world 9

testReadText2
Hello world 0
..
done...
```

We may notice that the read/write operations on the 100 byte blocks take about 8 ms.

2.9 Low power operation period with `lowPowerHandler()`

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

To start lets us study the following example.

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

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

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

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

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

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

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

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

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

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

```
  TimerInit( &wakeUp, onWakeUp );
  onSleep();
}
```

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

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

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

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

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

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

The complete code is given below:

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

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

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

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

```
    delay(50);
    Wire.begin(29,28);
    SHT21.begin();

    t=SHT21.getTemperature();
    h=SHT21.getHumidity();
    Serial.print("Humidity(%RH): ");
    Serial.print(h);
    Serial.print("     Temperature(C): ");
    Serial.println(t);
    dt=(int)((t-(int)t)*100.0);  dh=(int)((h-(int)h)*100.0);
    sprintf(buff,"T:%d.%d, H:%d.%d\n",(int)t,dt,(int)h,dh);
    Serial.println(buff);
    Wire.end();
    digitalWrite(Vext, HIGH);
}

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

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

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

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

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

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

To do

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

Lab 3 - 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, ..)

3.1 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();`

3.1.1 Sending data packet

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

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // 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                       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
#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)*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);
}
```

3.1.2 Sending data packet with low power management

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

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

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

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#define timetillsleep 1000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // 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                       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
#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);
}
```

3.2 LoRa - receiving the packets with battery state

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

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // Hz
#define TX_OUTPUT_POWER                             14        // 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
#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
..
```

3.3 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.

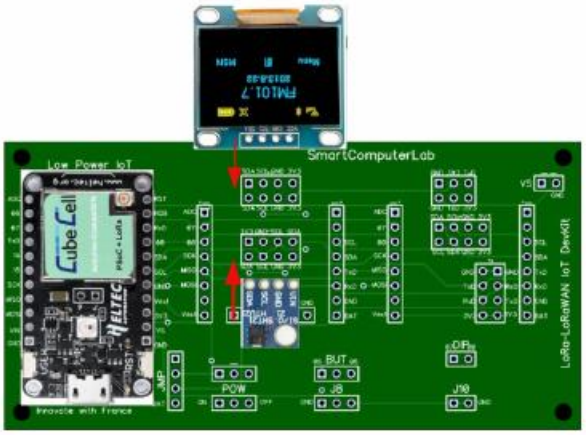


Fig 3.1 IoT DevKit for Low Power IoT design with main board and SHT21 sensor.

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

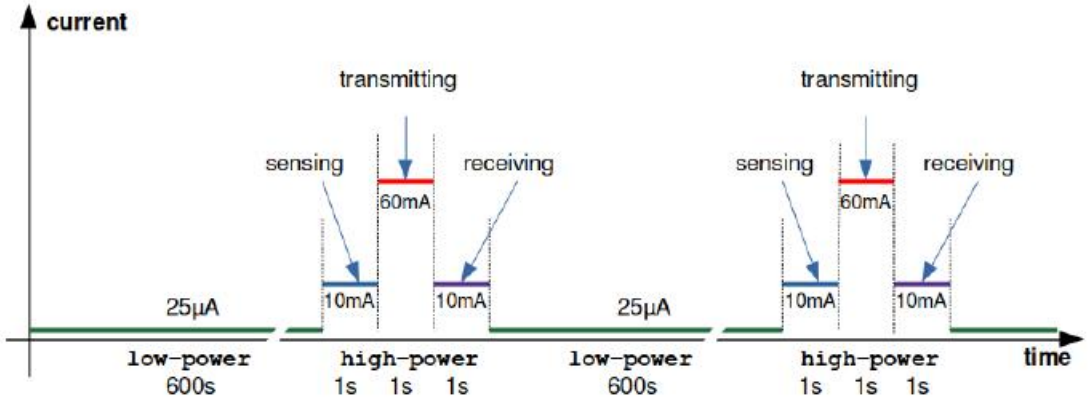


Fig 3.2 Power consumption diagram (current for 3.3V) with low and high power periods and high-power operational phases.

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

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

Fig 3.3 Transmission – airtime during high power period for 20-byte payload and 4/5 coderate

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

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

Let us calculate the average current consumption for :

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

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

For a battery with the capacity of 1000 mAh.

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

That is more than a year of operation.

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

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

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

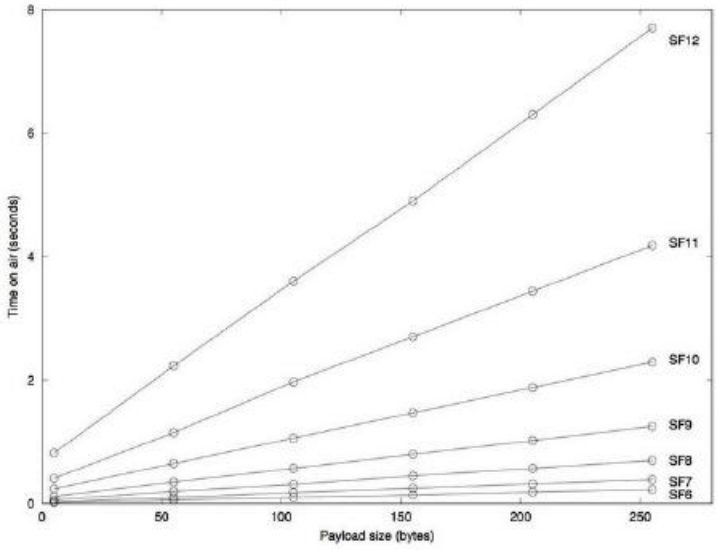


Fig 3.4 Time on Air for different spreading factors and payload size for 125 kHz band and 4/5 coding rate.

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

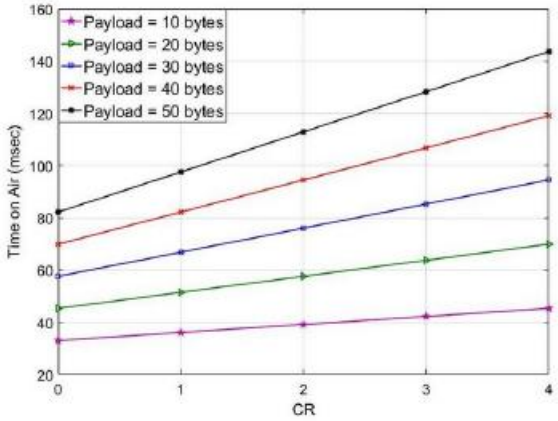


Fig 3.5 Time on Air for different payload size and coding rate (CR=1:5/4 to CR=4:8/4).

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

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

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

3.3.1 Terminal and the gateway codes

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

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

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

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

3.3.1.1 The code of terminal node

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#define timetillsleep 1000
#define timetillwakeup 15000
static TimerEvent_t sleep;
static TimerEvent_t wakeUp;
uint8_t lowpower=1;
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm up to 22, 20dBm - 100 mW
#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
#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);
}
```

3.3.1.2 The code of gateway (receiver) node

The corresponding gateway node receives the data sent by the terminal and sends an ACK packet with new **timeout** value generated randomly. The node has no capacity to relay the received data to the IoT server via an Internet connection. This kind of nodes are studied in the Lab 4.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
/*
 * set LoraWan_RGB to 1,the RGB active in loraWan
 * RGB red means sending;
 * RGB green means received done;
 */
#ifndef LoraWan_RGB
#define LoraWan_RGB 0
#endif
#define RF_FREQUENCY                                868000000 // 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                       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
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

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

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

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

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

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

long tout;

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

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

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

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

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

```
..
packet send
TX done!

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

packet send
TX done!

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

packet send
TX done!

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

packet send
TX done!

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

..
```

To do:

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

Lab 4 – 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 board to our kit and connect both MCU**s** via an UART link.

4.1 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.

4.1.1 Sending data from CubeCell via an UART link

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

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

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

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

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

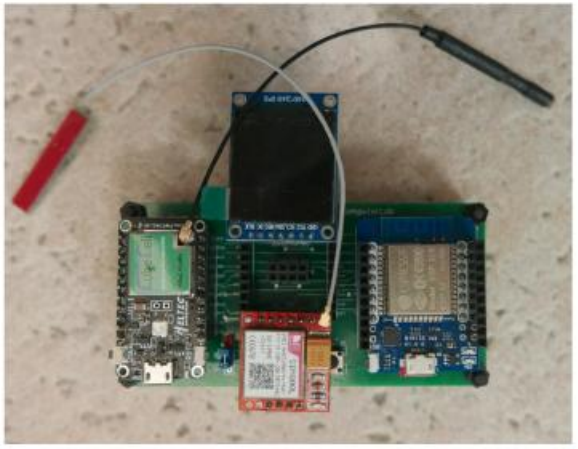
On the ESP32 board side we use **`HardwareSerial`**. The board receives the data on the HardwareSerial link and sends them to the ThingSpeak server. The ESP32 board is connected to ThingSpeak server via a WiFi link. The received (via **`uart`**) data are sent to the ThingSpeak server every 20 seconds.

```
#include <Arduino.h>
#include <ThingSpeak.h>
#include <WiFi.h>

HardwareSerial uart(2); // 16 -RX,  17-TX

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

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

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

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
```

```
  delay(2000);
  Serial.println();
  Serial.println("Goodnight moon!");
  // set the data rate for the SoftwareSerial port
  uart.begin(9600);  delay(2000);
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
}
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");Serial.println(ip);
ThingSpeak.begin(client);
}

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

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

4.2 LoRa receiver on CubeCell and WiFi gateway with ESP32

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



4.2.1 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                                868000000 // Hz
#define TX_OUTPUT_POWER                             20        // dBm - 25mW, 17-50mW, 20 - 100mW

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

4.2.2 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                                868000000 // 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                       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
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Define the payload size here

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

typedef enum
{
    LOWPOWER,
    RX,
    TX
} States_t;

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

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

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

long tout;

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

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

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

void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    Rssi=rssi;
    rxSize=size;
```

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

The UART data are received by the ESP32 and sent to the ThingSpeak server (see **4.1.2)**

4.3 ESP32 LoRa receiver and WiFi gateway 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.
This is the code 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    868E6
#define sf      8
#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(8);
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();
}
```

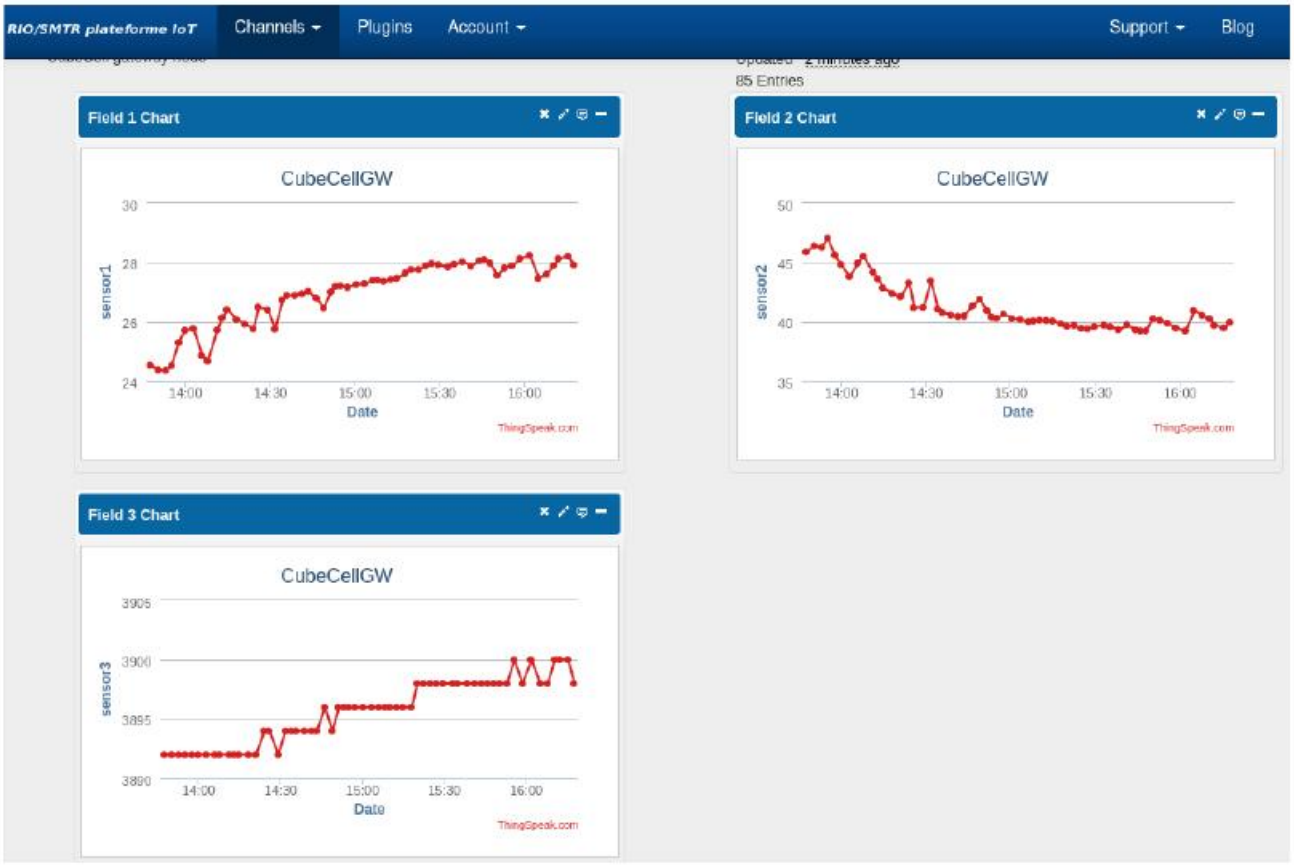


Fig 4.1 ThingSpeak charts - Temperature, humidity and battery level (node with solar cell).

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

To do:

- Add an external EEPROM to prepare the LoRa parameters. Write a program to store LoRa 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).

5. Assignment – terminal's scheduler in gateway node

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

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

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

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

Table des matières