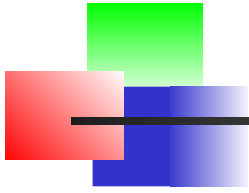


Low Power IoT Architectures

SmartComputerLab

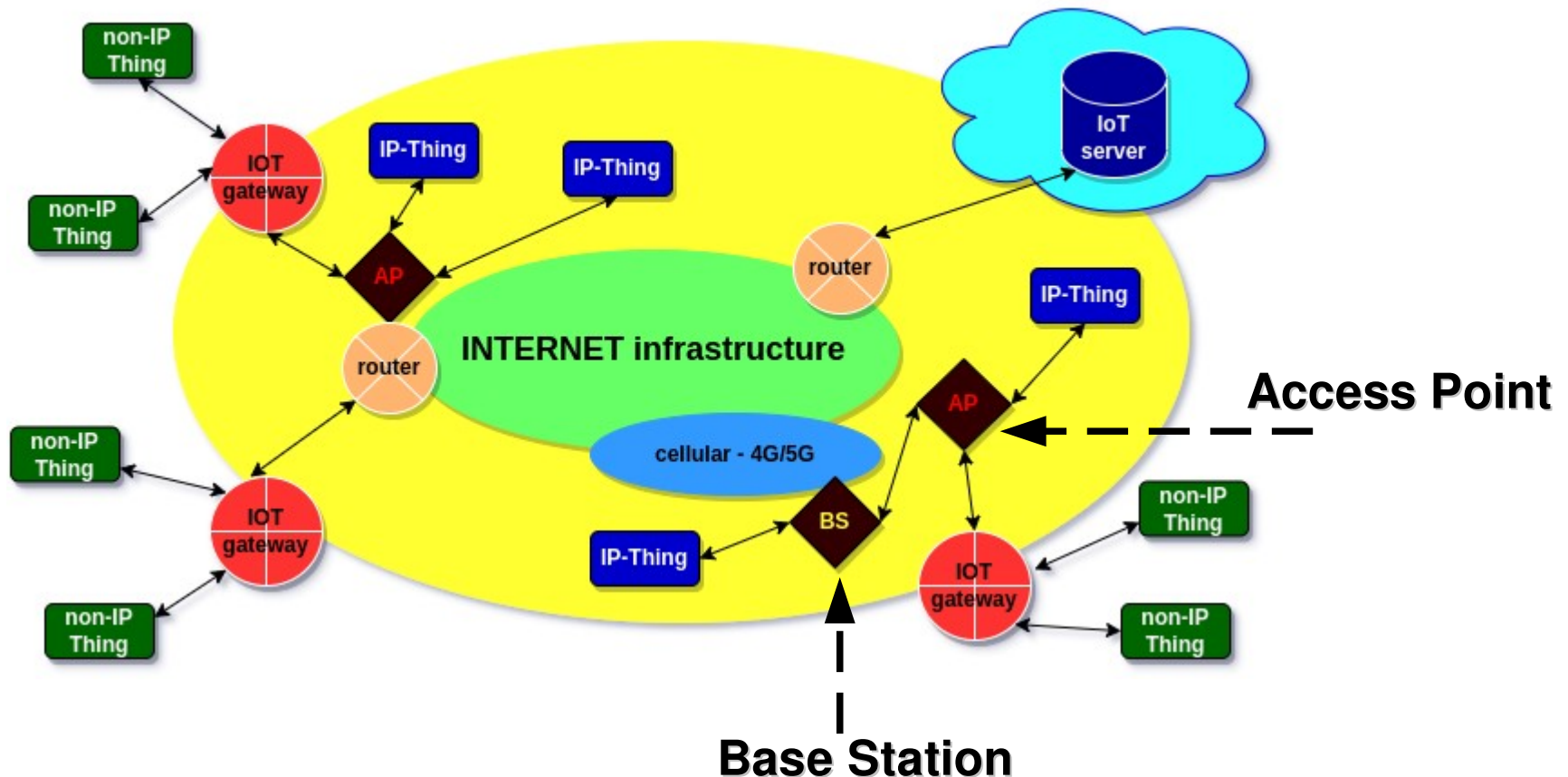


Low Power IoT Architectures

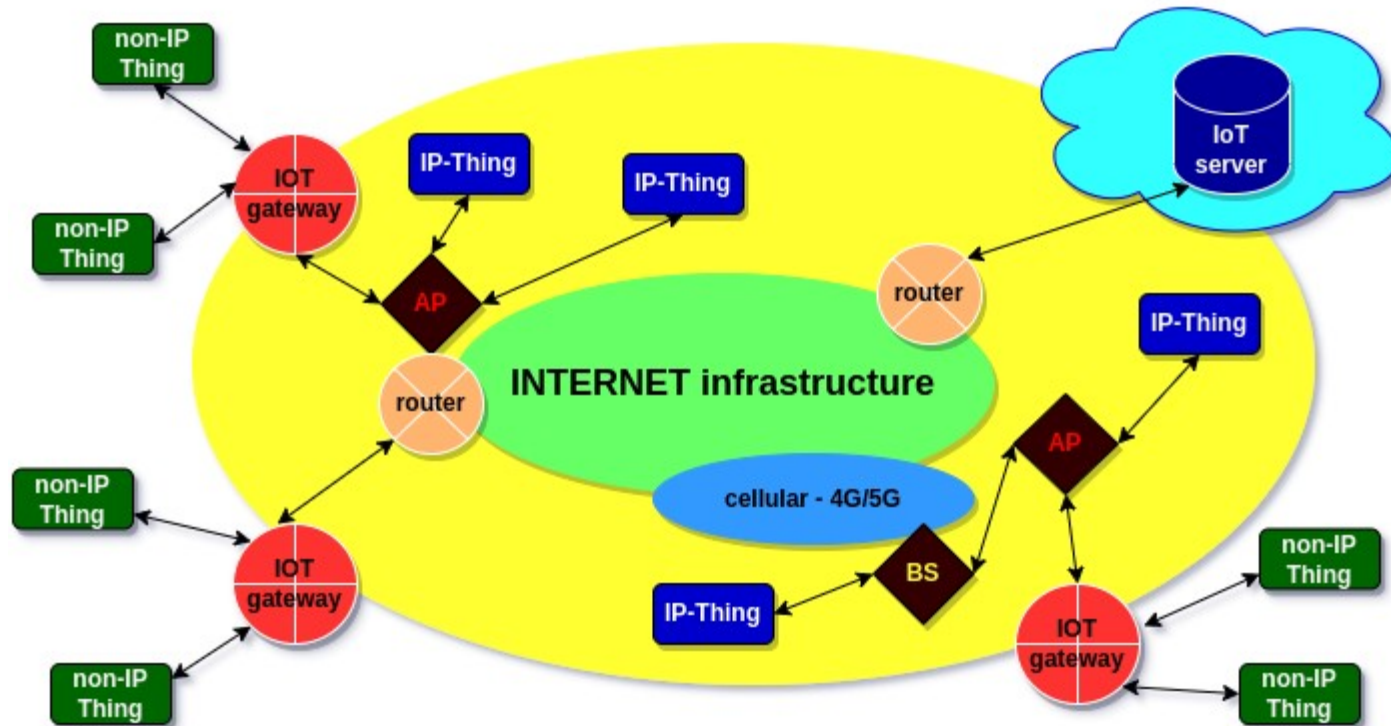
Low power IoT architectures are crucial for ensuring that IoT devices can operate efficiently with **minimal energy consumption**, especially in applications where devices are expected to function for extended periods on battery power or energy harvesting.

By focusing on **energy-efficient components**, **communication protocols**, and design practices, it is possible to develop IoT systems that meet the demands of modern applications while **minimizing their environmental impact**.

IoT Architectures



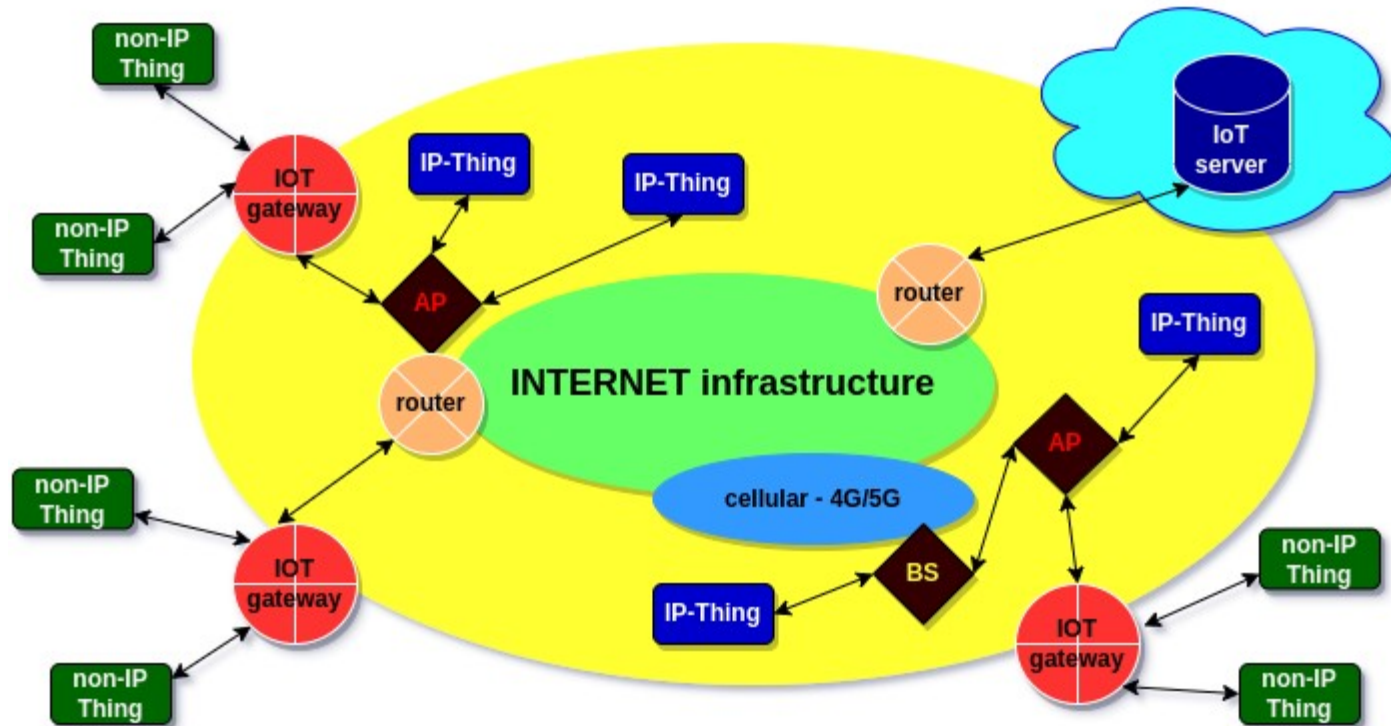
IoT Architectures



IP-Thing : Direct Terminal connected to Inet via WiFi/Ethernet link

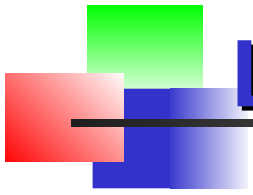
non-IP-Thing : Remote Terminal connected to Inet via LoRa link

IoT Architectures



IoT-Gateway : ex. LoRa – WiFi gateway

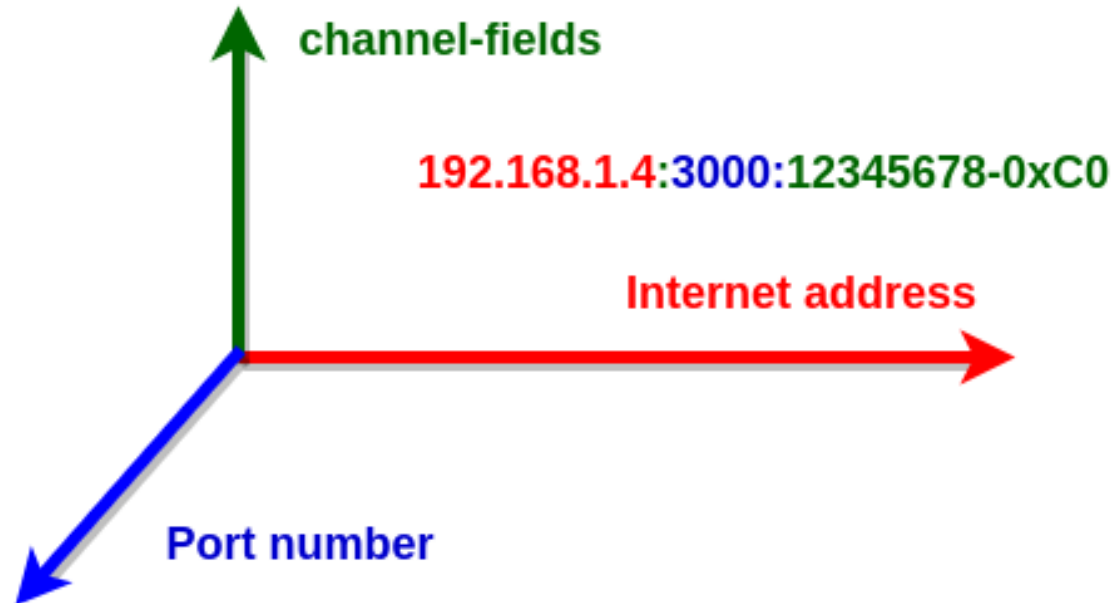
IoT-Server : ex. ThingSpeak



Low Power IoT Architectures

- IP Terminals (DT) and Non-IP Terminals (RT)
- **low_power** and **high_power** stages and **phases**
- Power consumption analysis of IP Terminals (DT)
- Power consumption analysis of Non-IP Terminals (RT)
- Complete IoT – Architectures :
with IoT Terminals – **Gateways** - **Servers**

Global IoT space and IoT Sockets



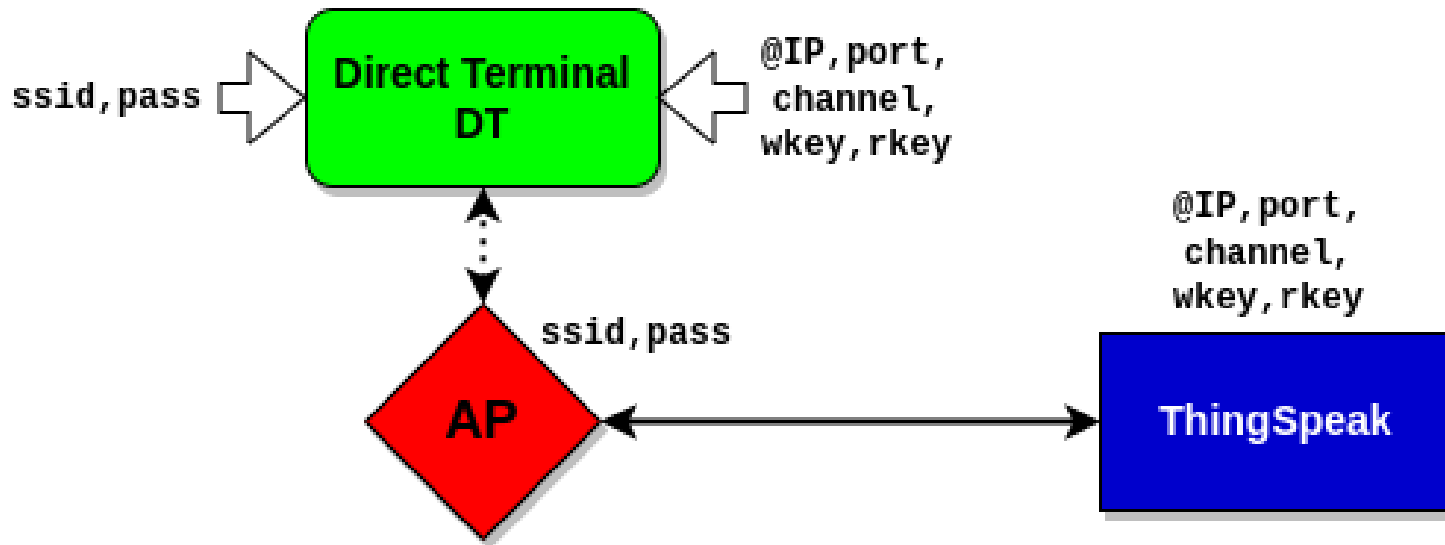
IoT socket => **IP address**: **Service port**: **Channel number-fields**

Direct Terminals know **IP address**: **Service port**: **Channel number**

Gateways know **IP address**: **Service port**

Remote Terminals know only **Channel number** (identifier)

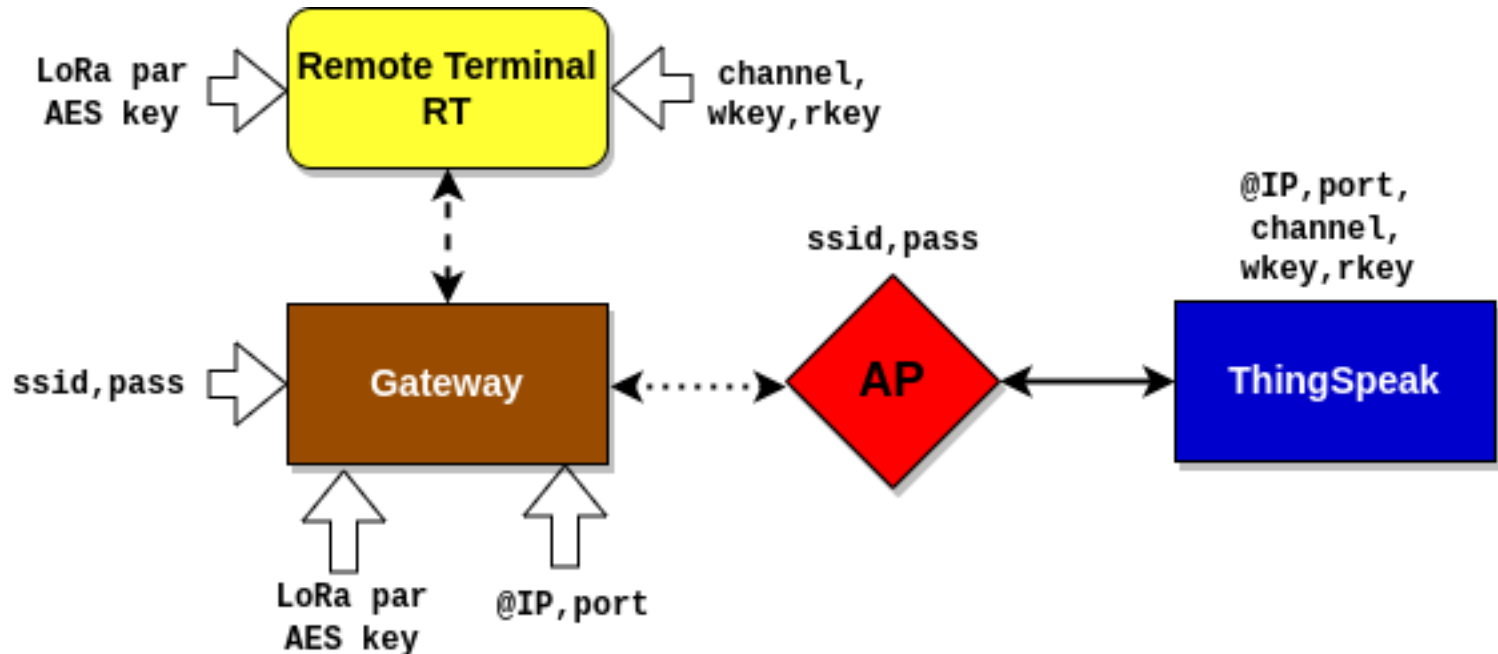
Direct Terminals and IoT Sockets



Direct Terminals know **IP address:Service port:Channel number**
plus: write and optionally read key

A channel contains fields (max.8) that may be interpreted as **IoT data streams**.

Global IoT space and IoT Sockets

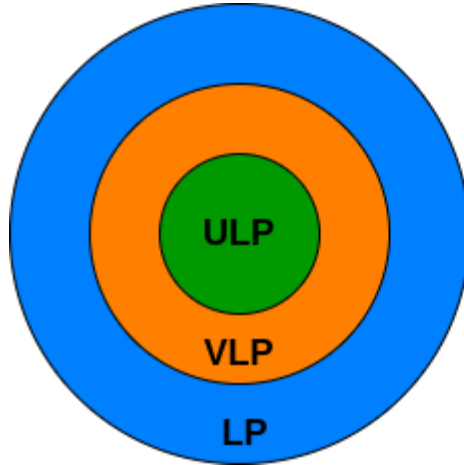


Gateways know **IP address:Service port**

Remote Terminals know only **Channel number** (identifier)

plus: write and optionally read key

Low and Very Low Power consumption IoT Architectures



ULP < 10 μ A

VLP < 100 μ A

LP < 1000 μ A

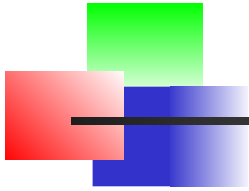
Example of average current (power) consumption:
deepsleep mode for **low_power** stage = 10 μ A and 100s
normal mode for **high_power** stage = 40mA and 0.5s

low_power charge + **high_power** charge = $10\mu\text{A} \cdot 100\text{s} + 40\,000\mu\text{A} \cdot 0.5\text{s} = 1000\mu\text{C} + 20000\mu\text{C} = 21\text{mC}$

$\text{average_current} = \text{charge/time} = 21\text{mC}/100.5\text{s} = 0.21\text{mA} = 210\mu\text{A}$

To do:

Calculate the same for **low_power** stage duration of 600s.



Terminals: Operational modes

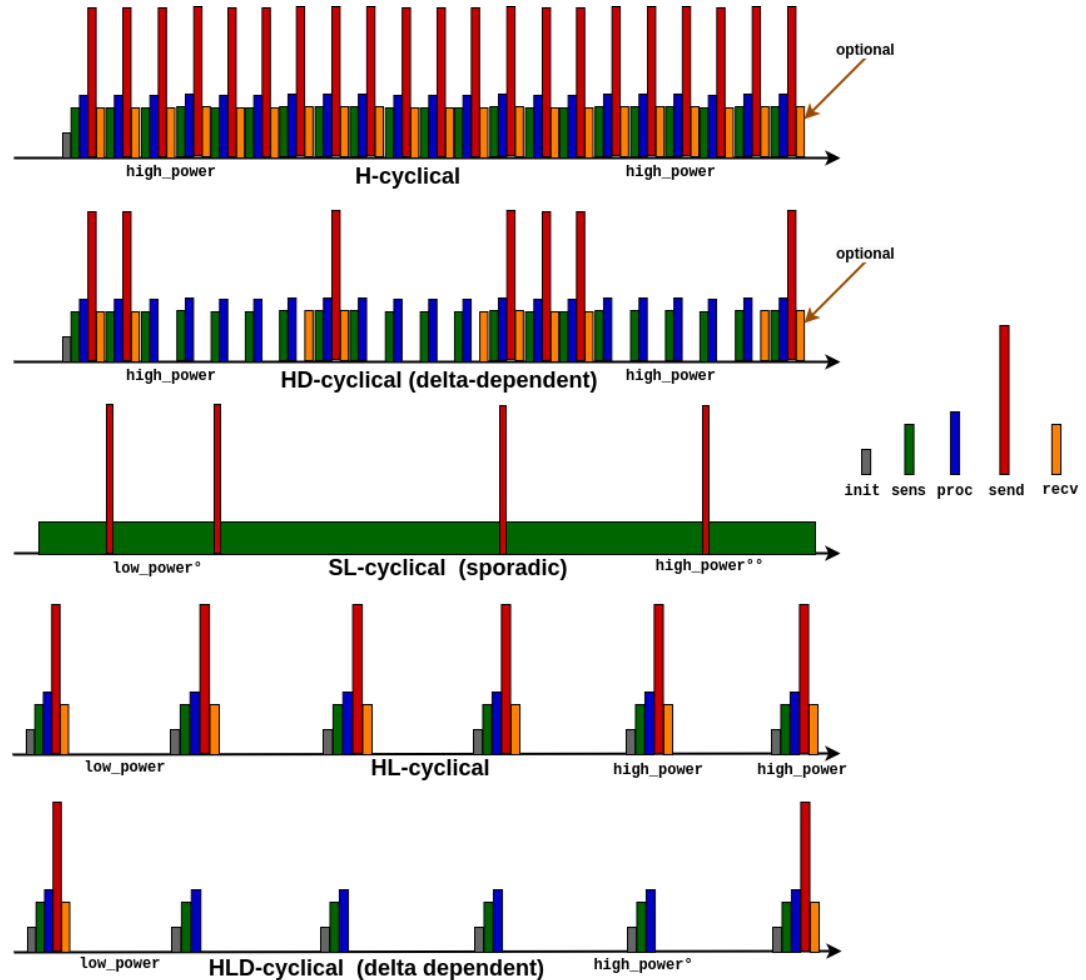
high average current

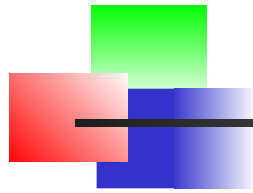


delta (δ) parameter
defines required
precision-difference

“sporadic cycle” –
activated by an
interruption (level
change) signal

low average current





high_power stage - phases

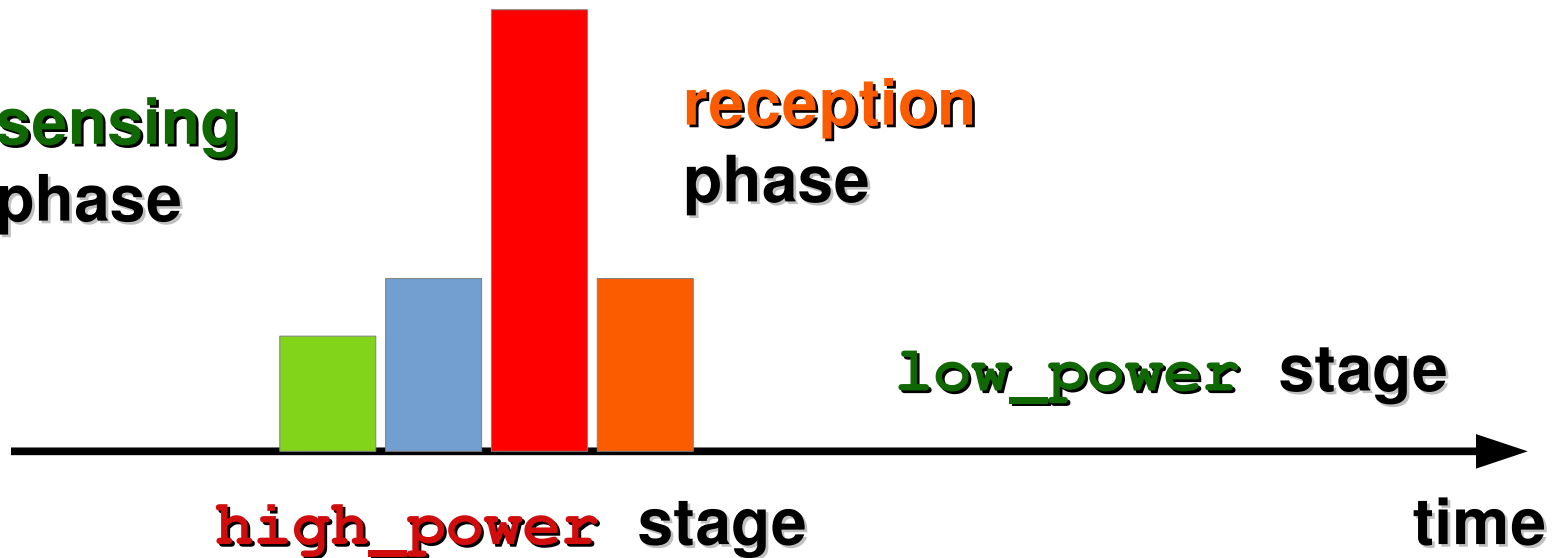
transmission phase

processing phase

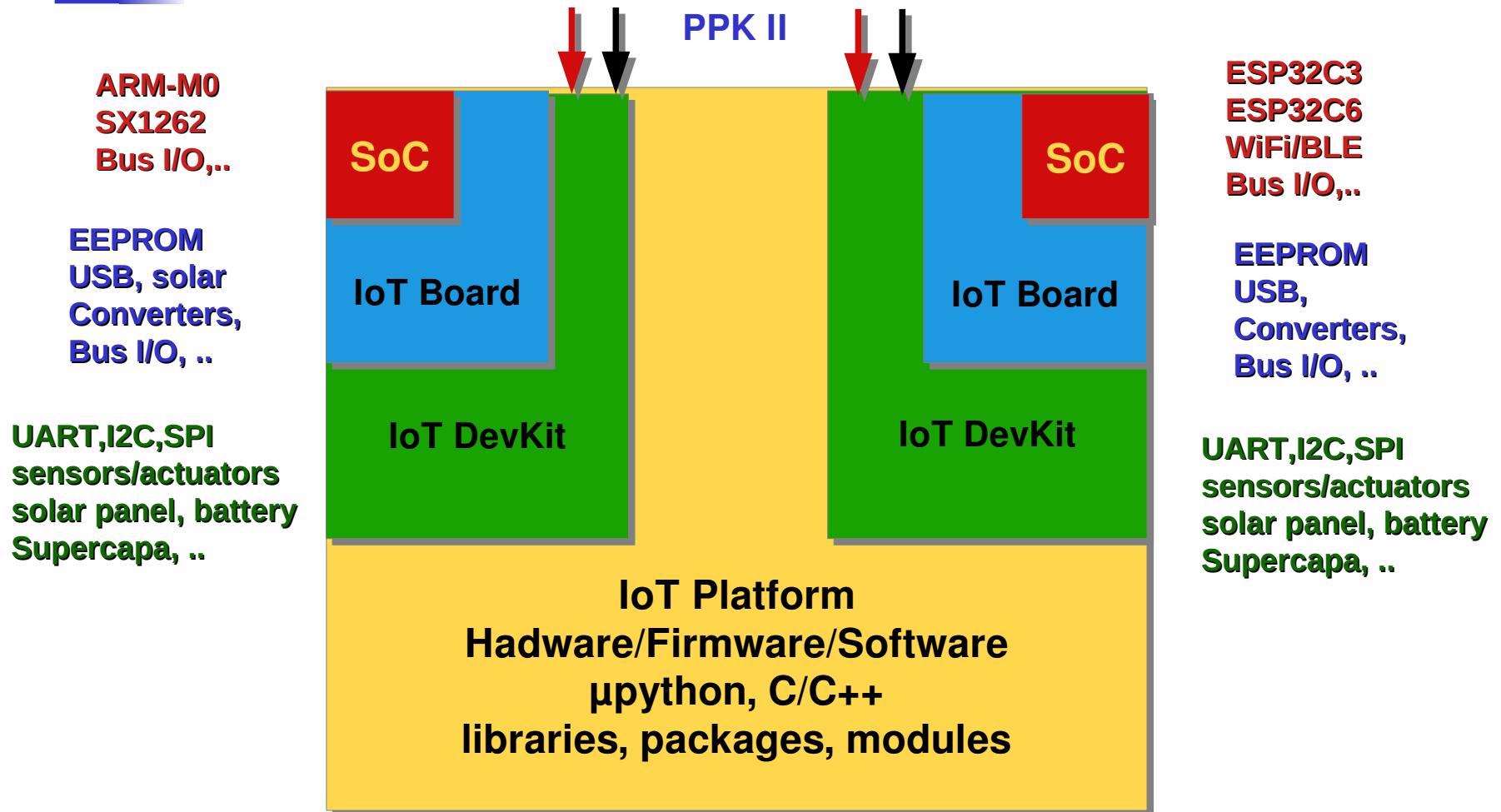
sensing phase

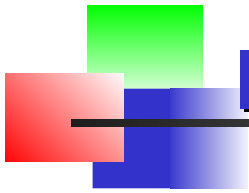
reception phase

low_power stage



From IoT SoC to IoT Platform



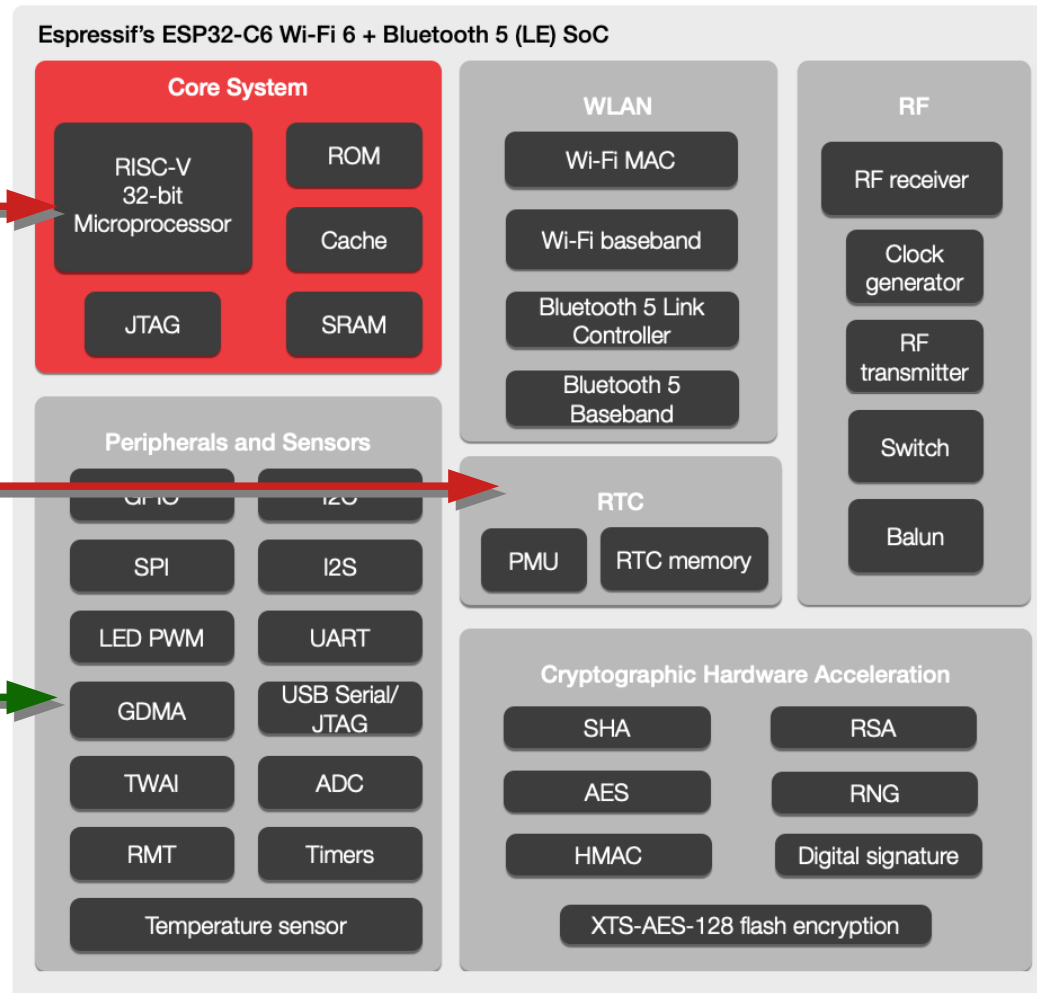


IoT SoC ESP32C6: low power features

RISC-V: RV32
5 stages pipeline

Low Power (2mA)
RISC-V: RV32
2 stages pipeline

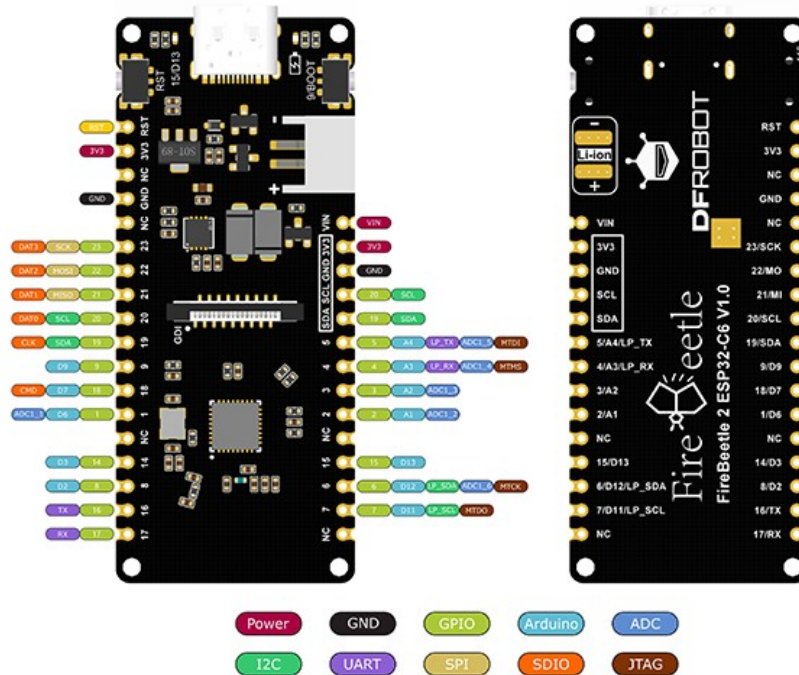
Buses including
Low Power
(I2C,UART)



WiFi6 with TWT -
Target Wake Time

Crypto and
Security

IoT Board (Direct Terminal)



IoT Board: DFRobot FireBeetle2 (ESP32C3) : EEPROM, battery, solar converters, USB bus, I2C, UART, SPI, .. (low/high power)

SmartComputerLab IoT DevKits

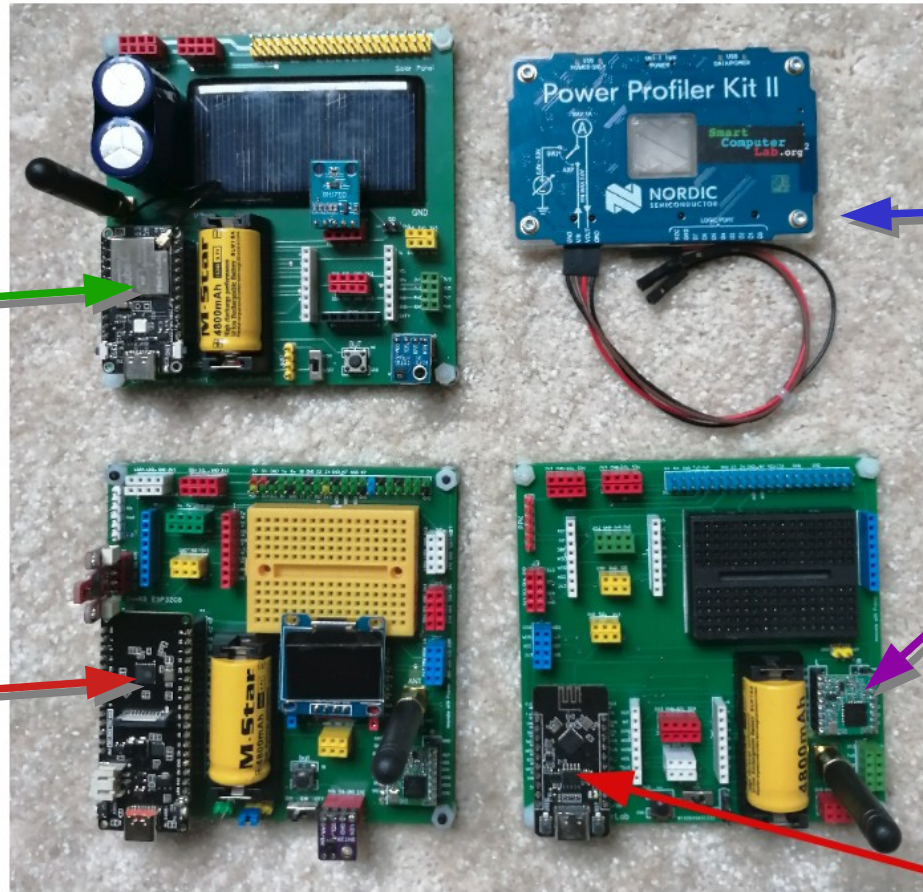
Heltec CubeCell
(ASR6501)

RFRobot
FireBeetle 2
(ESP32C6)

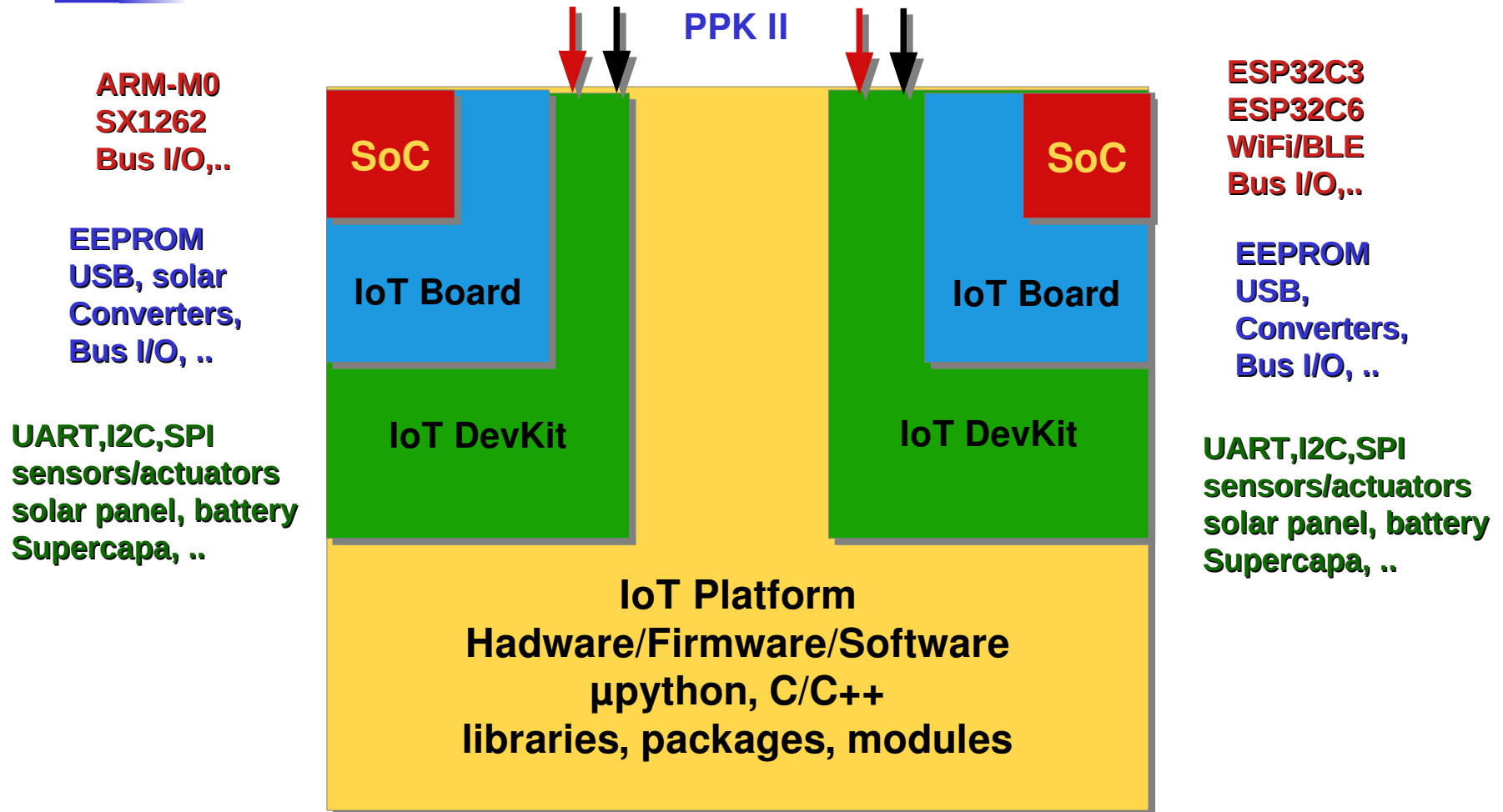
Nordic PPK II
(power
profiler kit)

SX1276 (LoRa
modem)

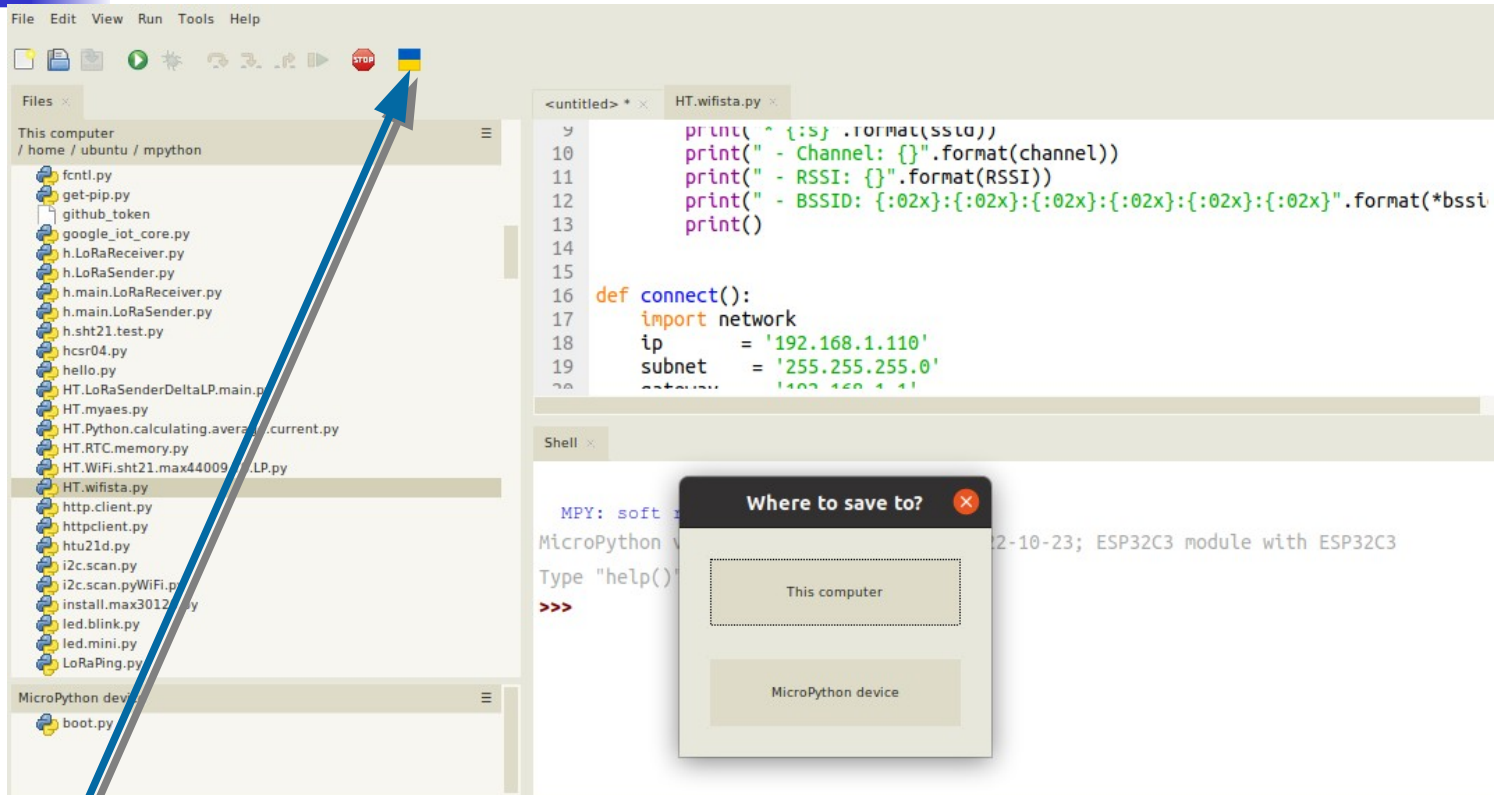
Heltec (ESP32-C3)



From IoT SoC to IoT Platform

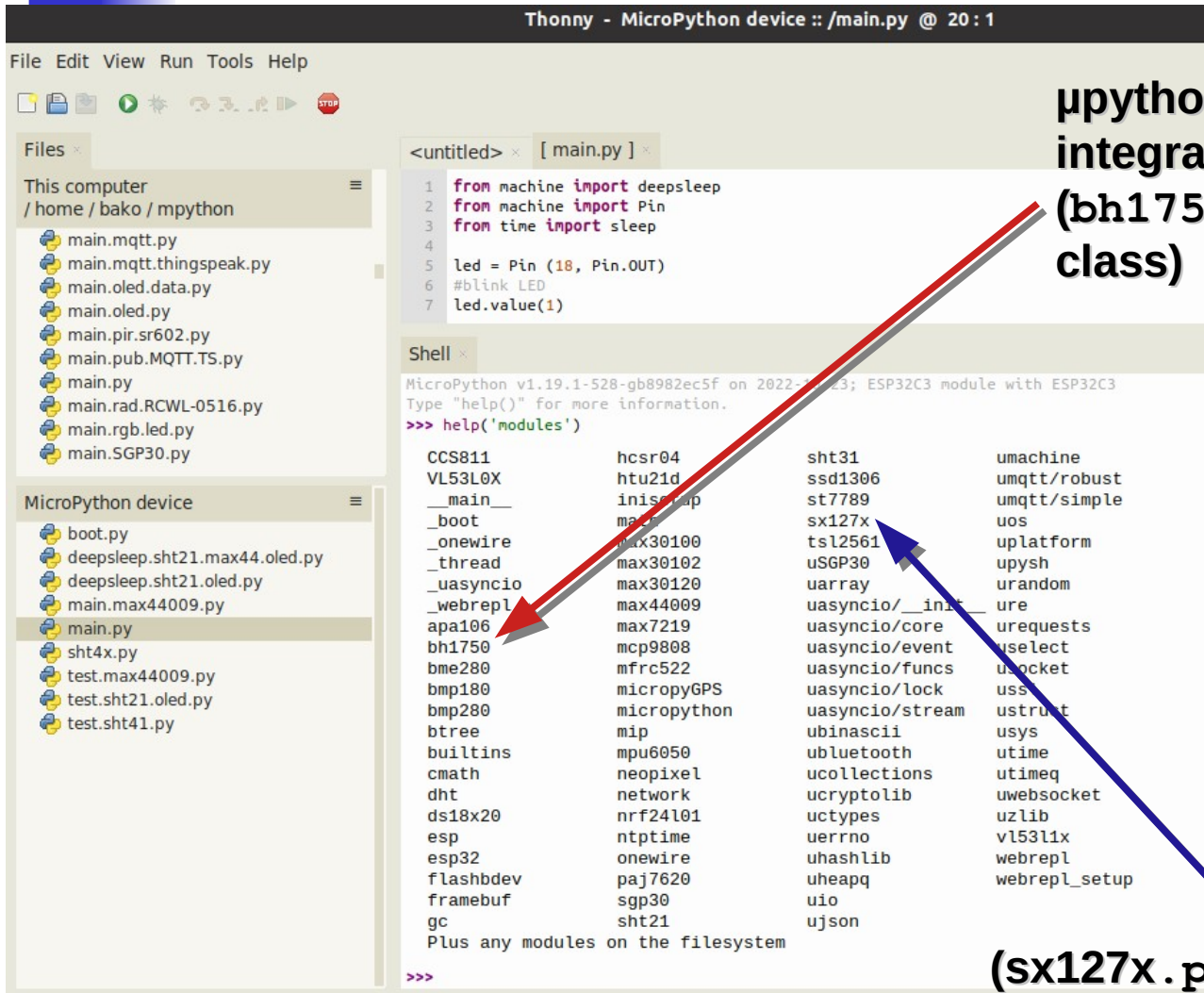


Thonny : IoT Platform -μPython



IoT Platform (examples):
Thonny IDE: microPython
Jupyter: microPython

Thonny : IoT Platform -µPython



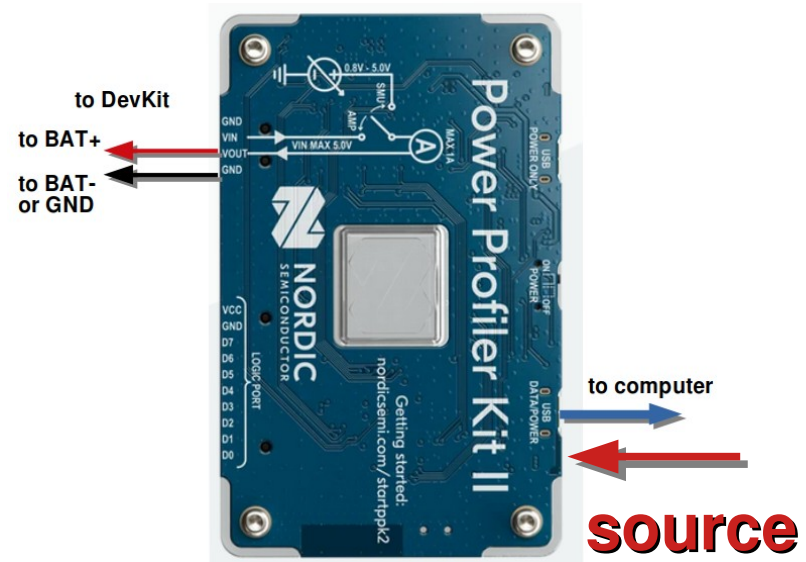
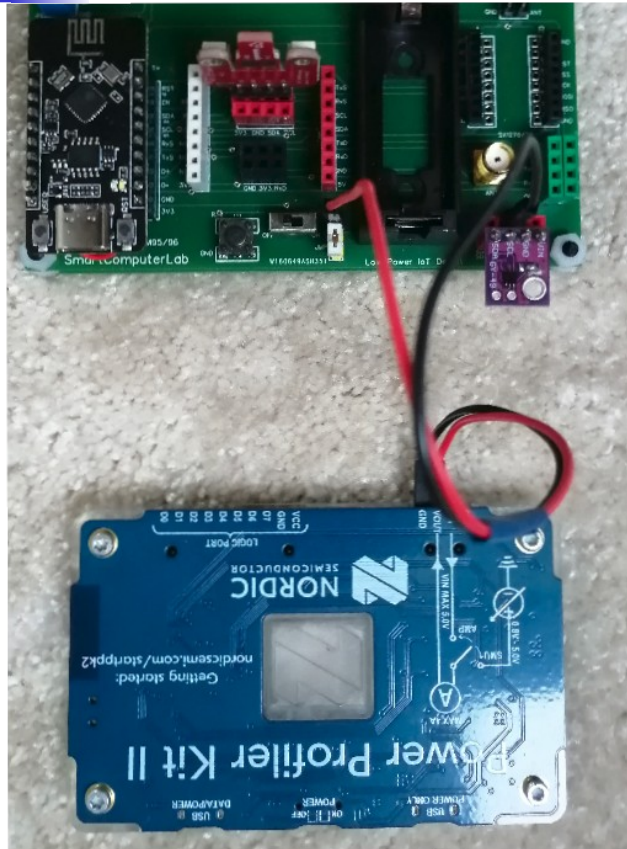
µpython interpreter and
integrated (cold) modules:
(bh1750.py with BH1750
class)

sensor driver

modem driver

(sx127x.py with SX127x class)

Power Profiler Kit II : connection



Power Profiler with **source** mode

Power Profiler Kit II - IDE

PPK2
F45048669DCD

MODE

Source meter **Ampere meter**

Set supply voltage to 3300 mV

Enable power output ☒

SAMPLING PARAMETERS

1,000 samples per second

Sample for 13 hours

Estimated RAM required 187.2 MB
1 ms period

Start

Clear session data

Show Minimap ☒

main window



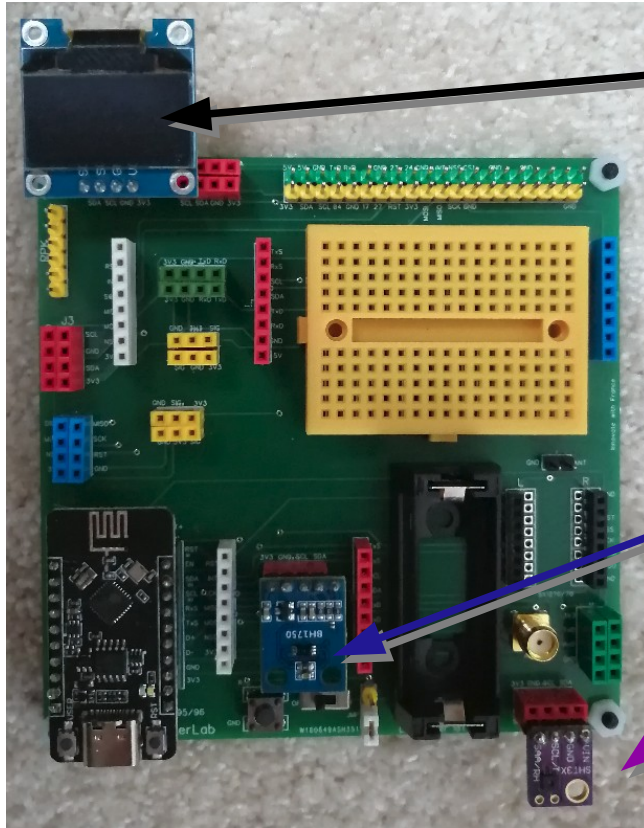
**source
mode**

low_power stage

high_power stage

**Average current consumption in
low_power stage (10.39µA)**

Example: HL cycle operation with sensors and OLED display



OLED display – **sdd1306**,
two sensors to capture
the **temperature**,
the **humidity**, and
the **luminosity** or **brightness**
values:

BH1750 (L) - luminosity

SHT31 (T/H) – temperature/humidity

Attention:

All these components
communicate over the same
(shared) I2C bus !



Example: HL cycle operation with sensors and OLED display

```
from machine import deepsleep
from machine import Pin, SoftI2C
from time import sleep
```

```
import ssd1306
import bh1750
import sht31
```

**Preparing buses,
drivers and sleep
operators**



Example: HL cycle operation with sensors and OLED display

```
def disp(p1,p2,p3):  
    oled = ssd1306.SSD1306_I2C(128, 64, i2c, 0x3c)  
    oled.fill(0)  
    oled.text("SmartComputerLab",0,0)    # column 0 and line 0  
    oled.text(p1,0,16)  
    oled.text(p2,0,32)  
    oled.text(p3,0,48)  
    oled.show()  
    sleep(5)  
    oled.poweroff()    # disconnects the OLED power lines
```

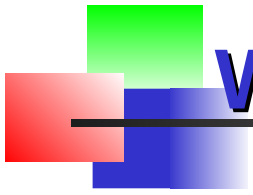
Defining display function to show the data during 5 seconds and disconnect

Example: HL cycle operation with sensors and OLED display



PPK diagram for the above example with two sensors and OLED screen operating with HL mode (high_power stage+low_power stage).

Note the current consumption in low_power stage – 20.42 μ A with three sensors/actuators attached.



What is ThingSpeak (.com)?

ThingSpeak is an **open-source software** written in **Ruby** which allows users to communicate with **internet enabled devices**.

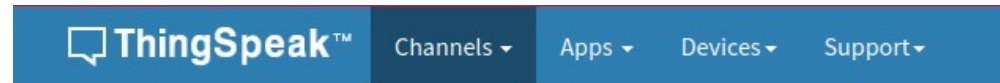
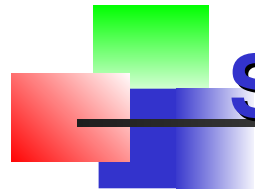
It facilitates data access, retrieval and **logging of data** by providing an API to both the devices and social network websites.

ThingSpeak was originally launched by **ioBridge** in 2010 as a service in support of IoT applications.

ThingSpeak has integrated support from the numerical computing software **MATLAB** from **MathWorks**, allowing **ThingSpeak.com** users to **store**, **analyze** and **visualize** uploaded data using MATLAB **without requiring the purchase of a MATLAB license from MathWorks**.

Attention: Free account on **ThingSpeak.com** is limited to last **256 records** (storage) and **usage frequency** of **max. 50 mHz** (communication).

Sending data to ThingSpeak



My Channels

New Channel

Search by tag

ThingSpeak.com – free account

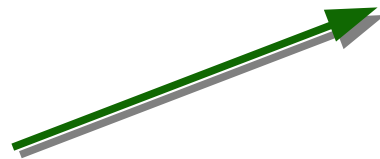
- **max. 10** channels
- **max. 256** last records per channel
- **max frequency ~50mHz**

Name ↕	Created ↕
Smart IoT 1 one, smartiotlabs Private Public Settings Sharing API Keys Data Import / Export	2021-10-16
Smart IoT 2 smartiotlabs, two Private Public Settings Sharing API Keys Data Import / Export	2022-01-05
Smart IoT 3 three, smartiotlabs Private Public Settings Sharing API Keys Data Import / Export	2022-04-07
Smart IoT 4 smartiotlabs, four Private Public Settings Sharing API Keys Data Import / Export	2022-04-07

Sending data to ThingSpeak

TS Channel ID
it will be **your device identifier**

TS Channel Fields → IoT data streams
are the places to store the received data
max 8 fields per channel
one field (**stream**) per sensor



ThingSpeak™ Channels Apps Devices Support

Smart IoT 1

Channel ID: 1538804
Author: mwa0000024358098
Access: Public

This channel is prepared DevKits identified by cha with LoRa-TS protocol.
one, smartiotlabs

Private View Public View Channel Settings Sharing API Keys

Channel Settings

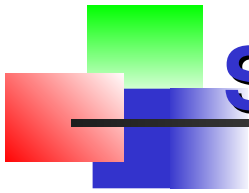
Percentage complete 70%

Channel ID 1538804

Name Smart IoT 1

Description This channel is prepared to use with Smart IoT DevKits identified by channel number. It operates

Field 1	Temperature	<input checked="" type="checkbox"/>
Field 2	Humidity	<input checked="" type="checkbox"/>
Field 3	Luminosity	<input checked="" type="checkbox"/>
Field 4	Battery state	<input checked="" type="checkbox"/>



Sending data to ThingSpeak



Smart IoT 1

Channel ID: 1538804
Author: [mwa0000024358098](#)
Access: Public

This channel is prepared to use with Smart IoT DevKits identified by channel number. It operates with LoRa-TS protocol.
 one, smartiotlabs

Private View Public View Channel Settings Sharing API Keys Data Import / Export

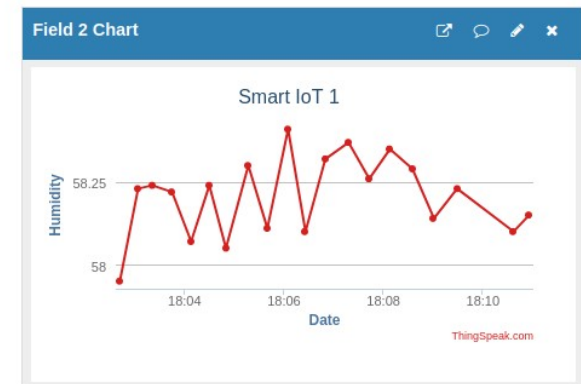
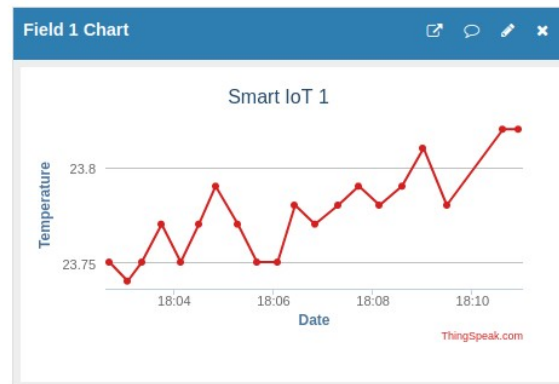
[+ Add Visualizations](#) [+ Add Widgets](#) [Export recent data](#)

[MATLAB Analysis](#) [MATLAB Visualiza](#)

Channel 1 of

Channel Stats

Created: [2 years ago](#)
Last entry: [3 months ago](#)
Entries: 714



Channel ID
it will be your
device identifier

One field (stream)
per sensor:

- Temperature
- Humidity
- Luminosity

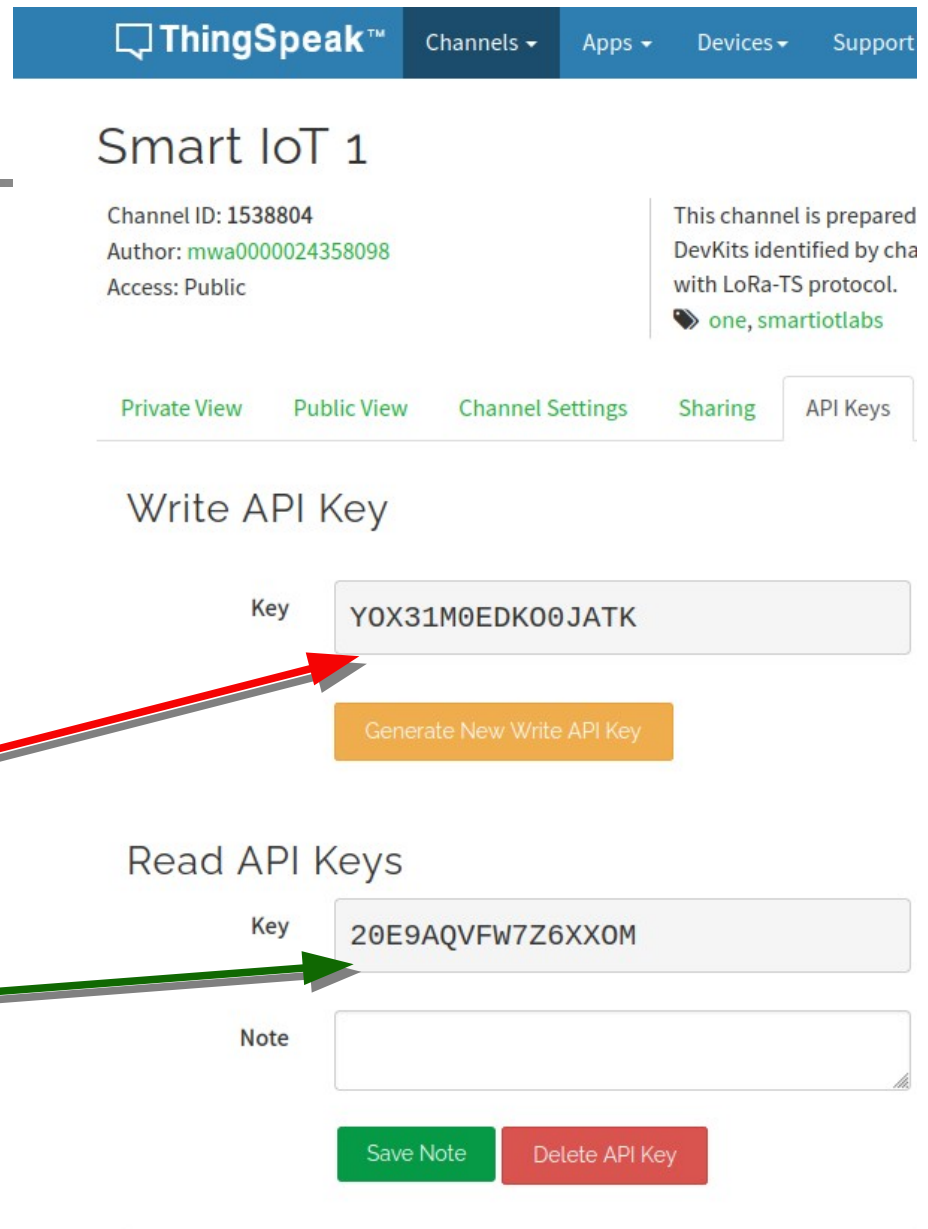
...

Sending data to ThingSpeak

Channel ID
it will be your device identifier

Write Key

Read Key



The screenshot shows the ThingSpeak interface for a channel named "Smart IoT 1". The channel ID is 1538804, the author is mwa0000024358098, and the access is public. A note mentions that the channel is prepared for DevKits identified by "cha" with the LoRa-TS protocol, linked to "one, smartiotlabs". Navigation tabs include Private View, Public View, Channel Settings, Sharing, and API Keys. The "API Keys" tab is active, showing a "Write API Key" section with a key field containing "Y0X31M0EDK00JATK" and a "Generate New Write API Key" button. Below this is a "Read API Keys" section with a key field containing "20E9AQVFW7Z6XX0M", a "Note" field, and buttons for "Save Note" and "Delete API Key".

ThingSpeak™ Channels Apps Devices Support

Smart IoT 1

Channel ID: 1538804
Author: mwa0000024358098
Access: Public

This channel is prepared DevKits identified by cha with LoRa-TS protocol.
one, smartiotlabs

Private View Public View Channel Settings Sharing API Keys

Write API Key

Key Y0X31M0EDK00JATK

Generate New Write API Key

Read API Keys

Key 20E9AQVFW7Z6XX0M

Note

Save Note Delete API Key



Example: sending sensor data to ThingSpeak via WiFi

```
def connect() :
    import network
    ssid      = "PhoneAP"
    password  = "smartcomputerlab"
    station = network.WLAN(network.STA_IF)
    if station.isconnected() == True:
        print("Already connected")
        return station

    station.active(True)
    station.connect(ssid,password)
    station.config(txpower=8.5)
    while station.isconnected() == False:
        pass
    print("Connection successful")
    print(station.ifconfig())
    return station

def disconnect() :
    import network
    station = network.WLAN(network.STA_IF)
    station.disconnect()
    station.active(False)

#disconnect()
#connect()
```

```
# to be uncommented for test
# to be uncommented for test
```

Preparing module
wifista.py for
communication via WiFi
– with personal AP



Example: sending sensor data to ThingSpeak via WiFi

```
import machine
from machine import deepsleep, Pin, SoftI2C
import max44009
import sht21
import wifista
import thingspeak
from thingspeak import ThingSpeakAPI, Channel, ProtoHTTP

i2c = SoftI2C(scl=Pin(9), sda=Pin(8), freq=100000)
sensor = max44009.MAX44009(i2c)
luminosity=sensor.lux
temperature = sht21.SHT21_TEMPERATURE(i2c)
humidity = sht21.SHT21_HUMIDITE(i2c)
print("current temperature: " +str(temperature))
```

Reading data from sensors : **sensing** phase



Example: sending sensor data to ThingSpeak via WiFi

```
print("current temperature: " +str(temperature))
rtc = machine.RTC()
stfloat=0.0
r=rtc.memory()
print('woken with value',r)    # testing the last temp value
if(r!=b' '):                    # skiping first empty value
    stfloat=float(r)
    print(stfloat)

if (temperature>(stfloat+0.2) or temperature<(stfloat-0.2)):
# delta in °C ex. 0.2 °C
```

processing data from sensors : **processing** phase with **delta** parameter. Comparing with previously sent data.

Attention: requires the use of RTC memory



Example: sending sensor data to ThingSpeak via WiFi

```
if (temperature>(stfloat+0.2) or temperature<(stfloat-0.2)):
    rtc.memory(str(temperature))
    channel_living_room = "1538804"
    field_temperature = "Temperature"
    field_humidity = "Humidity"
    field_luminosity = "Luminosity"
    active_channel = channel_living_room
    thing_speak = ThingSpeakAPI([
        Channel(channel_living_room , 'YOX31M0EDKO0JATK',[field_temperature,
            field_humidity, field_luminosity])),protocol_class=ProtoHTTP,log=True)
    wifista.connect()
    thing_speak.send(active_channel, { field_temperature: temperature,
        field_humidity: humidity, field_luminosity: luminosity })
    print('send to TS')
    wifista.disconnect()

deepsleep(20000)  # or thing_speak.free_api_delay
```

**Sending data to ThingSpeak via WiFi
connection: **transmission** phase**

Example: sending sensor data to ThingSpeak via WiFi

Connection successful

('192.168.43.36', '255.255.255.0', '192.168.43.1', '192.168.43.1')

current temperature: 22.73444

woken with value b'22.82024'

22.82024

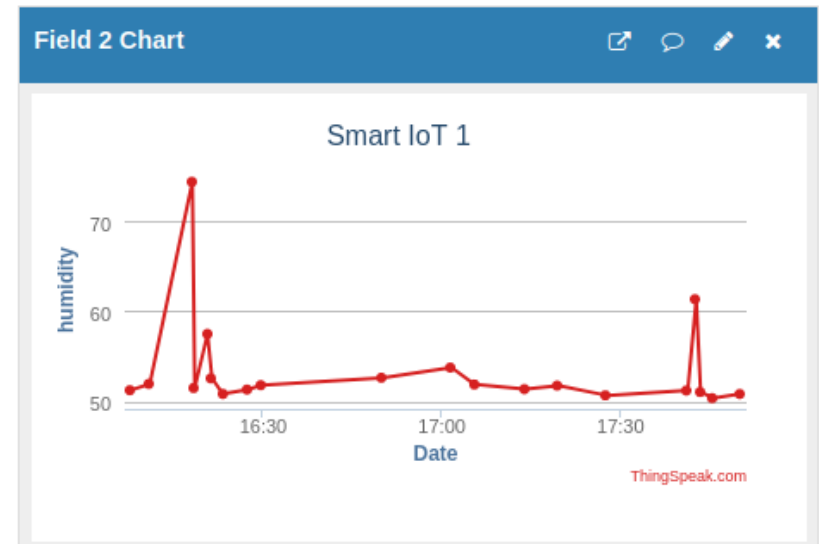
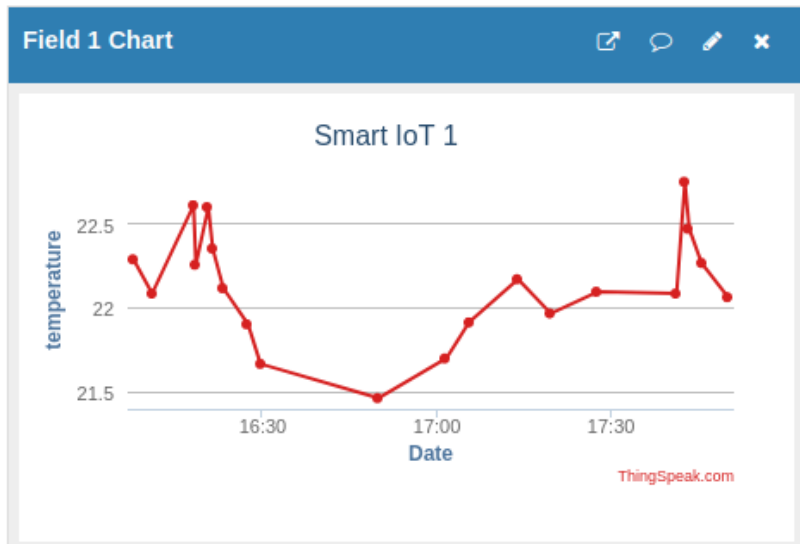
new value to rtc memory b'22.73444'

Already connected

ThingSpeak at 52.201.163.117:80

1538804 {'Luminosity': 348.48, 'Temperature': 22.73444, 'Humidity':

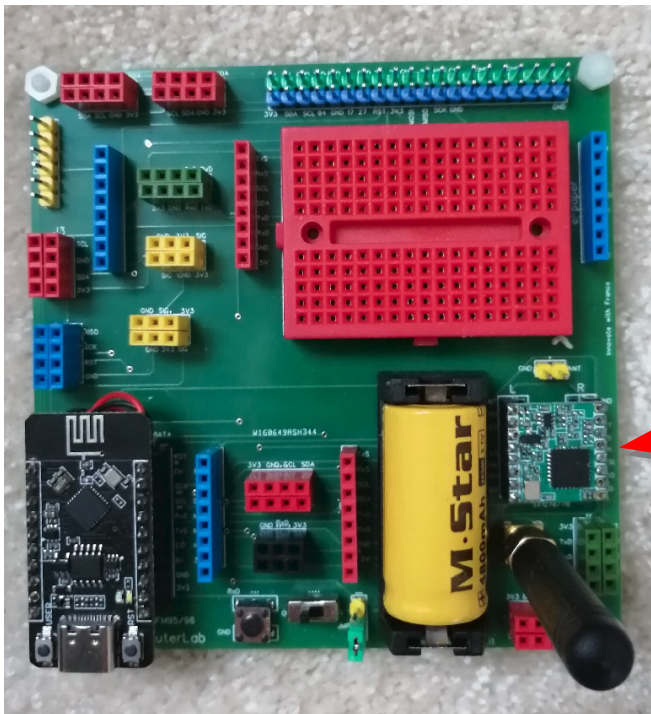
49.95398} #31, took 1.16s, next in 14.84s



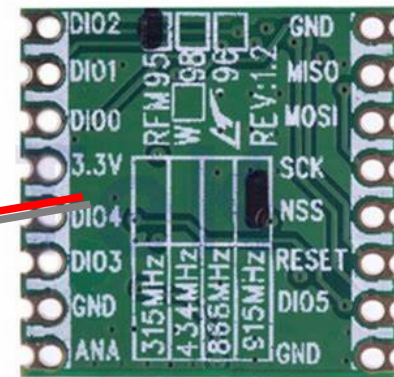
Part II: Long Range (LoRa) & Remote Terminals

Remote Terminals communicate with the IoT servers via the dedicated Gateways that relay the LoRa packets to the WiFi packets to be sent to IoT server.

To build a simple IoT architecture with Remote Terminals (at least one) we need to enlarge/enhance our initial Direct Terminal to a LoRa-WiFi Gateway.

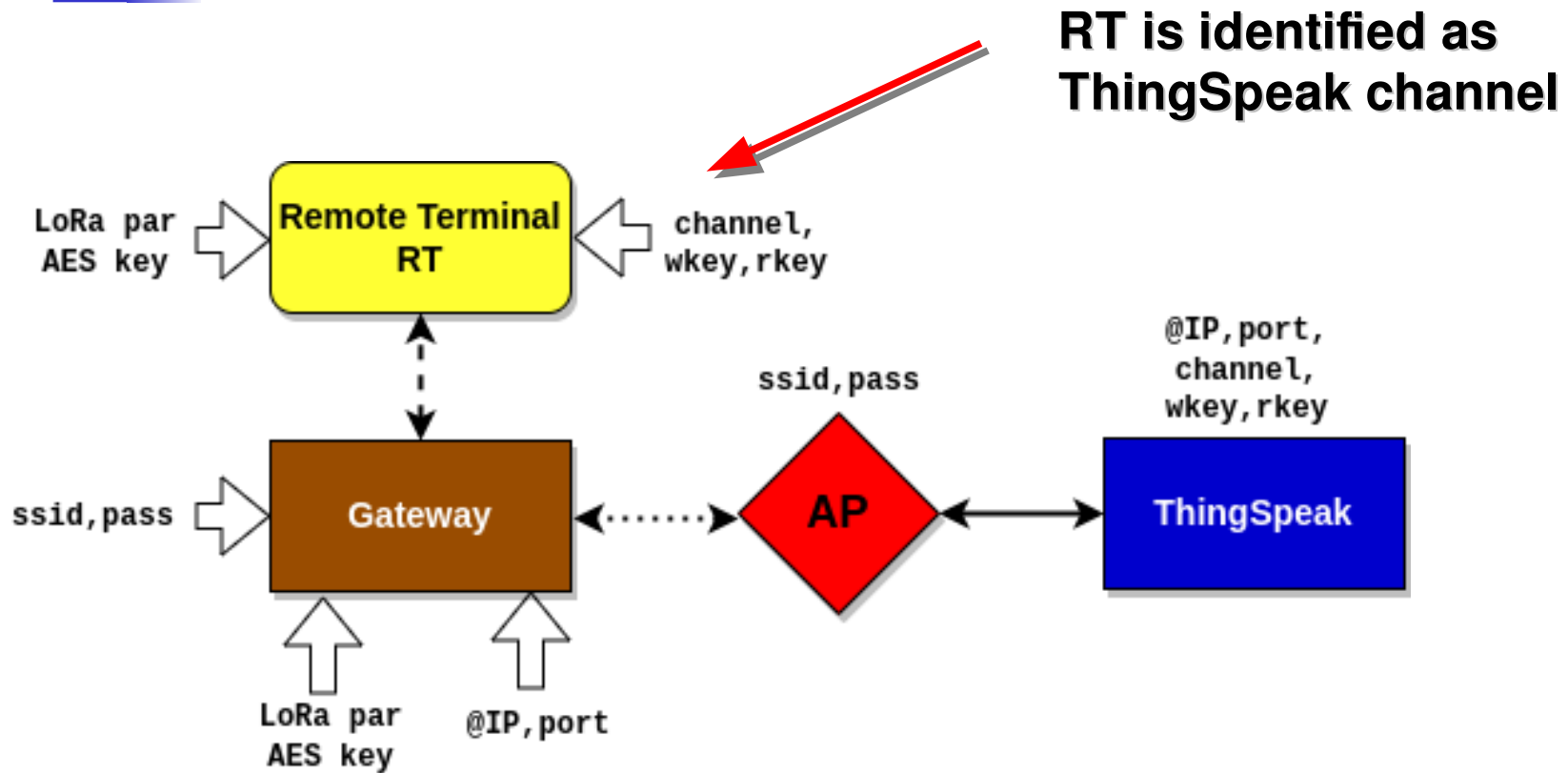


Remote Terminals or
Lora-WiFi Gateway



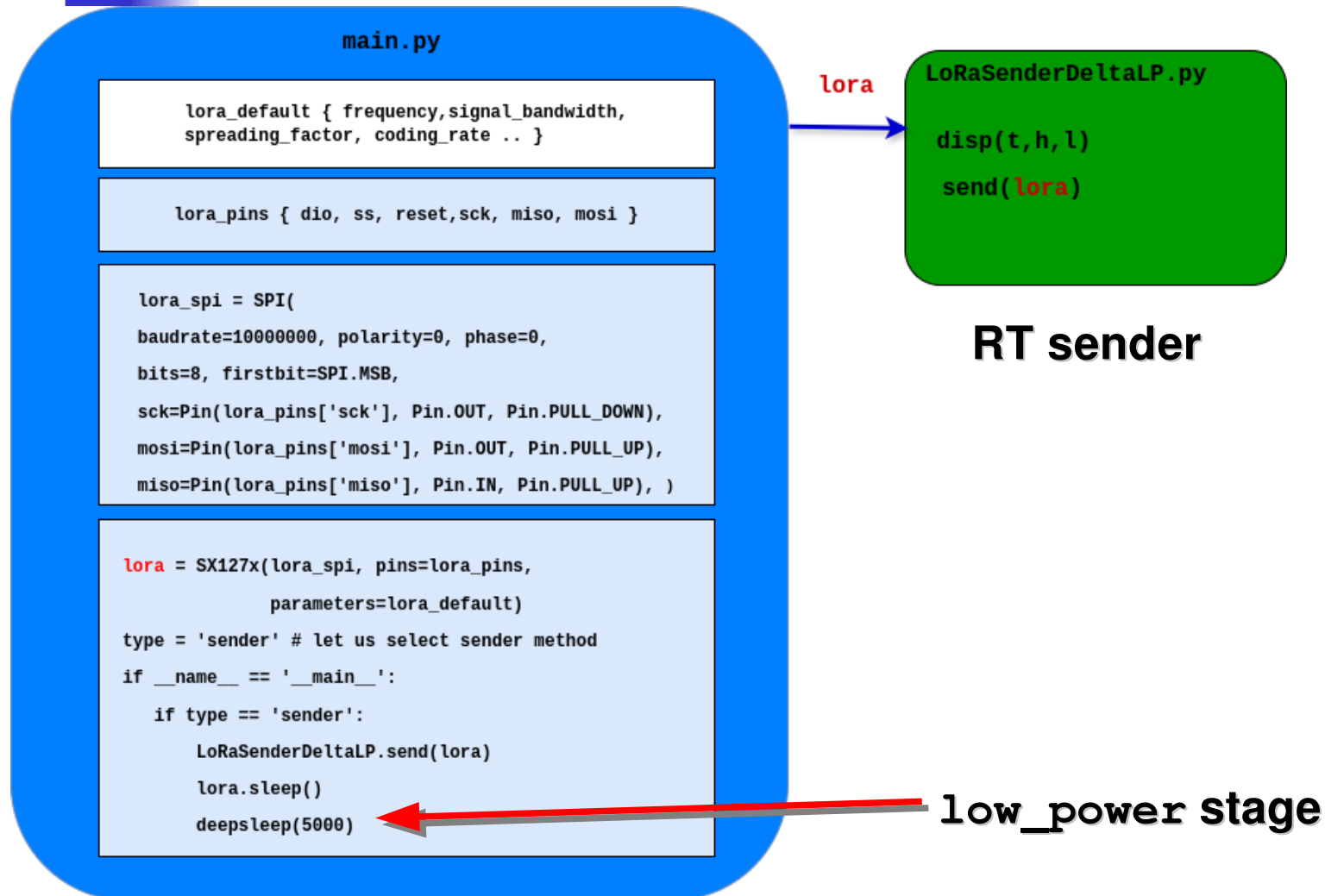
RFM95 – SX1278 LoRa
modem with SPI bus

Remote Terminal & Gateway (LoRa)



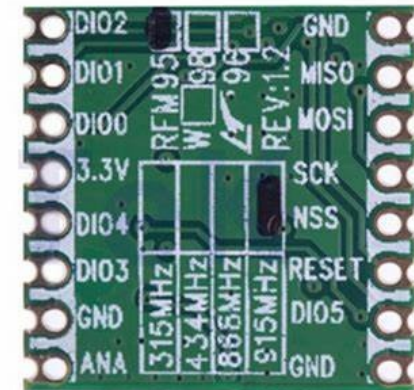
Complete IoT architecture with RT, GW and SV (ThingSpeak)

RT – LoRa SPI and radio parameters



RT – main and sender modules

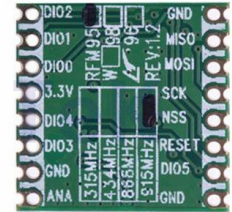
```
from machine import Pin, SPI
from sx127x import SX127x
from time import sleep
import LoRaSenderDelatLP
```



```
lora_pins = {
    'dio_0':2,          # interrupt signal - useful for callback
    'ss':4,             # 16 on SPI-LoRa ext. card
    'reset':10,         # necessary to start
    'sck':6,            # SPI bus clock signal
    'miso':5,           # SPI bus - master-in-slave-out
    'mosi':7,           # SPI bus - master-out-slave-in
}
```

Defining SPI connection lines to RFM95/SX1278 LoRa modem

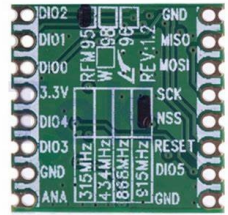
RT – main and sender modules



```
lora_default = {  
    'frequency': 869e6,                # base frequency: 8695e5,435e6, ..  
    'frequency_offset': 0,              # max 100mW  
    'tx_power_level': 14,               # modulation frequency band: 250e3,500e3  
    'signal_bandwidth': 125e3,          # spreading factor: 7,...,11  
    'spreading_factor': 9,              # coding rate: 5,...,8 (5/4,...,8/4)  
    'coding_rate': 5,                   #  
    'preamble_length': 8,               #  
    'implicitHeader': False,            # examples: 0x12 private, 0x34 public, ..  
    'sync_word': 0x12,                  #  
    'enable_CRC': False,                # normal: up-chirp, inverted: down-chirp  
    'invert_IQ': False,                  #  
    'debug': False,  
}
```

LoRa radio – default parameters

RT – main and sender modules



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

type = 'sender'      # let us select sender method

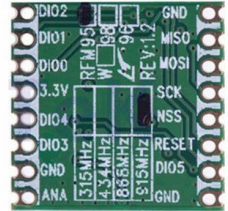
if __name__ == '__main__':
    if type == 'sender':
        LoRaSenderDeltaLP.send(lora)
        lora.sleep()
        deepsleep(5000)
```

Selecting LoRaSenderDeltaLP.py module with lora parameters

lora.sleep suspends LoRa modem activity

deepsleep MCU enters low_power stage

RT – main and sender modules



```
rtc = machine.RTC()
stfloat=0.0
r=rtc.memory()
print('woken with value',r)
if(r!=b'') :
    stfloat=float(r)
    print(stfloat)
# testing the last temp value
# skipping first empty value
```

```
chan =12345
wkey="abcdefghijklmnop"
lumsensor = max44009.MAX44009(i2c)
luminosity=lumsensor.lux
temperature = sht21.SHT21_TEMPERATURE(i2c)
humidity = sht21.SHT21_HUMIDITE(i2c)
```

high_power stage – **sensing** phase

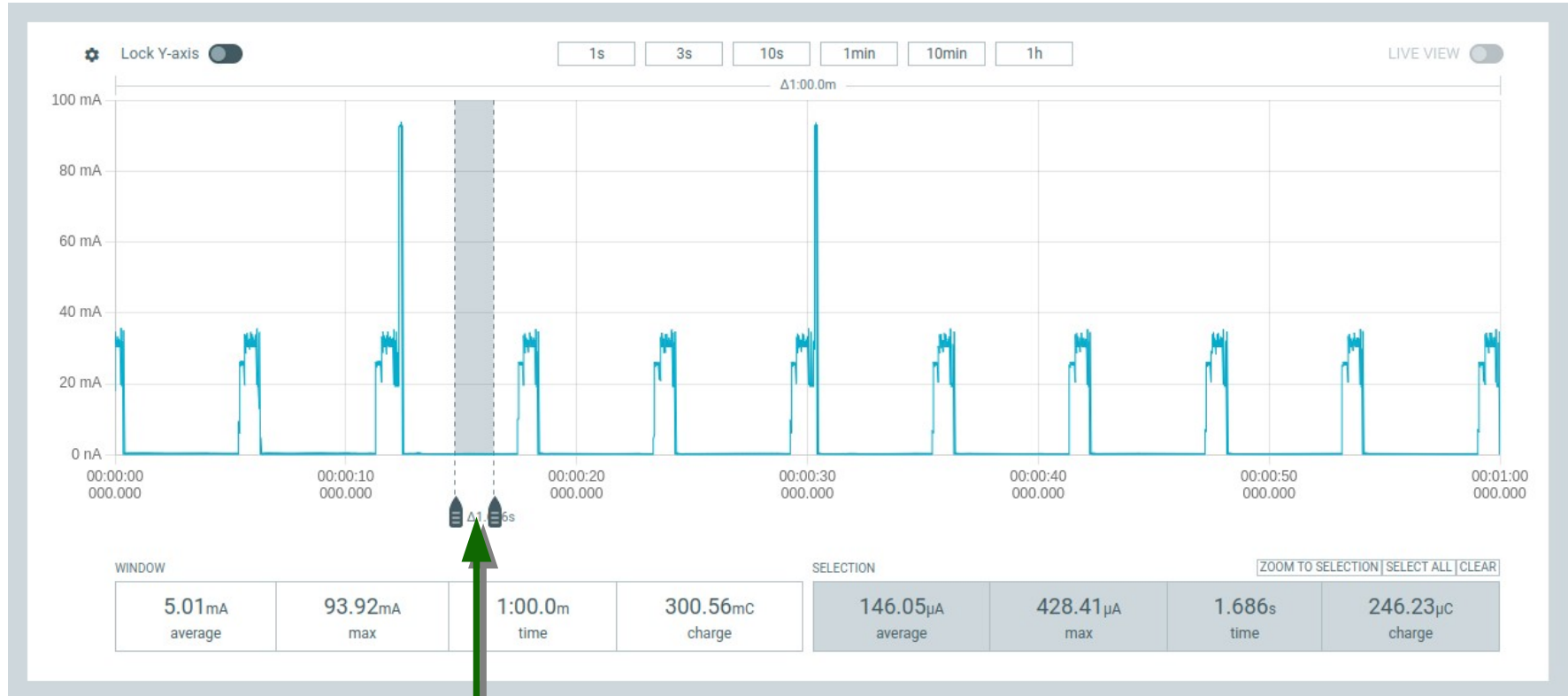
Building packet structure and sending packet

```
if (temperature>(stfloat+0.01) or temperature<(stfloat-0.01)):  
    led = Pin (18, Pin.OUT)  
    led.value(1)  
    rtc.memory(str(temperature))  
    print('new value to rtc memory',rtc.memory()) # test of the stored value  
    print("LoRa Sender")  
    counter=0.0  
    data=ustruct.pack('i16s4f',chan,wkey,  
                        temperature,humidity,luminosity,counter)  
    print("datalen: " + str(len(data)))  
    lora.beginPacket()  
    lora.write(data)  
    lora.endPacket()  
    sleep(1)  
    led.value(0)  
    disp(temperature,humidity,luminosity)
```



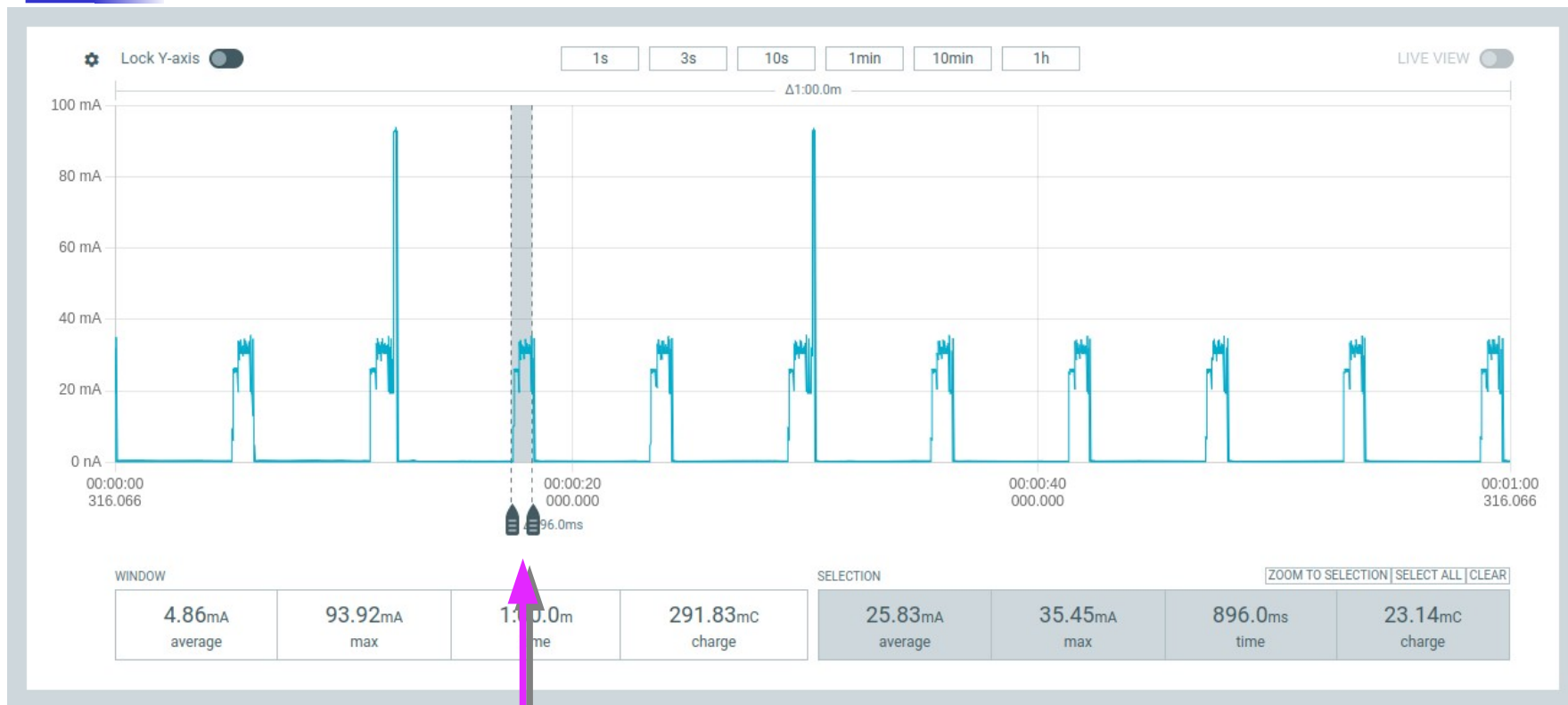
i 16s 4f
int chan char wkey[16] float temp,humi,lumi,count

RT - Power consumption with LoRa link



low_power stage – 146.05μA

RT - Power consumption with LoRa link



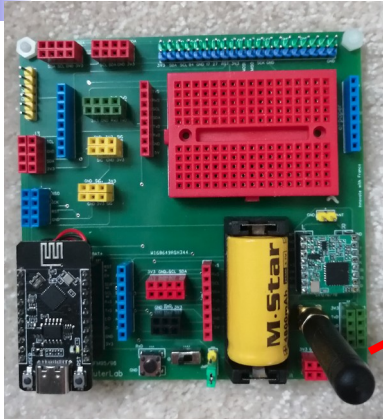
high_power stage sensing/processing phases only – 23.14mC

RT - Power consumption with LoRa link

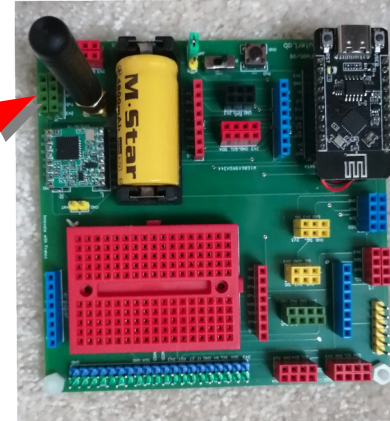


high_power stage sensing, processing, transmission phases –
39.67mC

LoRa-WiFi Gateway



LoRa



WiFi

chan	wkey	temp	humi	lumi	count
------	------	------	------	------	-------

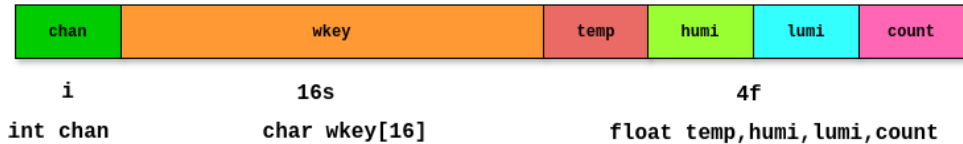
i 16s 4f
int chan char wkey[16] float temp,humi,lumi,count

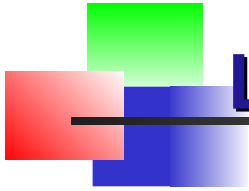
```
lora = SX127x(lora_spi, pins=lora_pins, parameters=lora_default)
type = 'receiver_gateway'                      # let us select sender method
```

```
if __name__ == '__main__':
    if type == 'sender':
        LoRaSenderDeltaTS.send(lora)
    if type == 'receiver':
        LoRaReceiver.receive(lora)
    if type == 'receiver_gateway':
        LoRaReceiverWiFiTS.receive(lora)
```

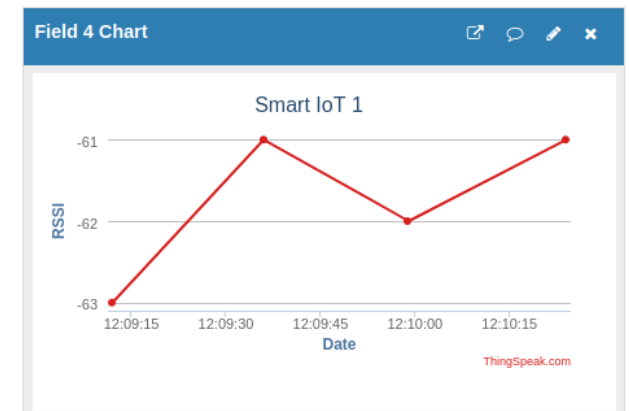
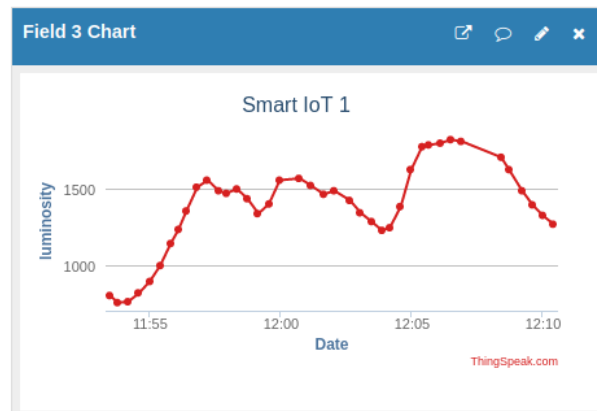
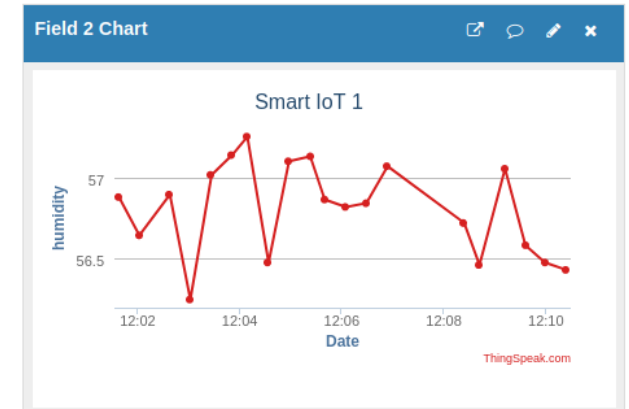
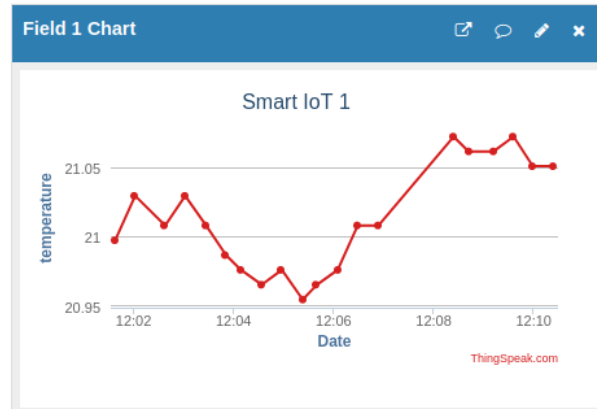

LoRa receive and WiFi-TS send

```
def receive(lora):
    print("LoRa Receiver and WiFi Gateway")
    wifista.connect()
    while True:
        if lora.receivedPacket():
            try:
                data = lora.readPayload()
                rssi = float(lora.packetRssi())
                chan, wkey, temp, humi, lumi, count = ustruct.unpack('i16s4f', data)
                disp(temp, humi, lumi, rssi)
                channel_living_room = chan
                active_channel = channel_living_room
                field_temp = "Temperature"
                field_humi = "Humidity"
                field_lumi = "Luminosity"
                field_rssi = "RSSI"
                thing_speak = ThingSpeakAPI([
                    Channel(channel_living_room, wkey, [field_temp, field_humi, field_lumi,
                                                            field_rssi])],
                    protocol_class=ProtoHTTP, log=True)
                thing_speak.send(active_channel, {field_temp: temp,
                                                  field_humi: humi, field_lumi: lumi, field_rssi: rssi})
            except Exception as e:
                print(e)
```





LoRa-WiFi Gateway

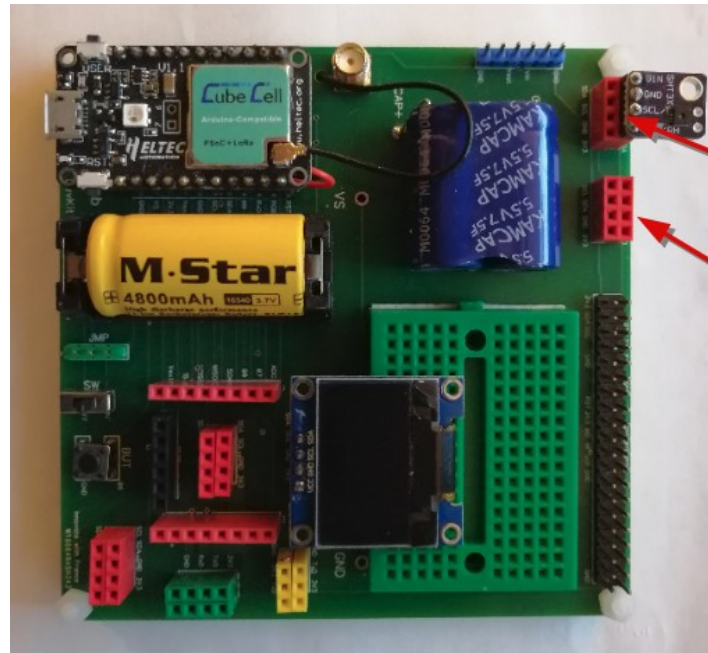


Part III : Long Range (LoRa) with CubeCell

The Remote Terminal on our HT board (RISC-V,WiFi,+LoRa modem) can operate in `low_power` mode that consumes about **150 μ A**. This value may be acceptable for **Low Power** solutions, however is too high for **Very Low Power** consumption. We need here different board specifically designed for LoRa communication such as CubeCell.

CubeCell is based on ARM-M0+**SX1262** LoRa modem

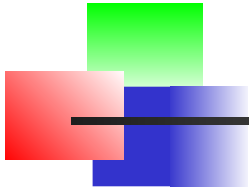
CubeCell **cannot be programmed with μ Python** – no space and no sufficient processing power



SDA		3V3
SCL		GND
GND		SCL
3V3		SDA

SCL		3V3
SDA		SCL
GND		SDA
3V3		GND

Attention !
The bus lines may be in different order !



CubeCell : Arduino IDE

Legacy IDE (1.8.X)

The screenshot shows the Arduino IDE 1.8.19 download page. On the left, there's a section with the Arduino logo, the version number '1.8.19', a description of the IDE as open-source software for writing and uploading code to Arduino boards, a link to the documentation, and a 'SOURCE CODE' section mentioning GitHub. On the right, a teal sidebar titled 'DOWNLOAD OPTIONS' lists download links for Windows (Win 7 and newer, Win 8.1 or 10 with a 'Get' button), Linux (32 bits, 64 bits, ARM 32 bits, ARM 64 bits), and Mac OS X (10.10 or newer). It also includes links for 'Release Notes' and 'Checksums (sha512)'.

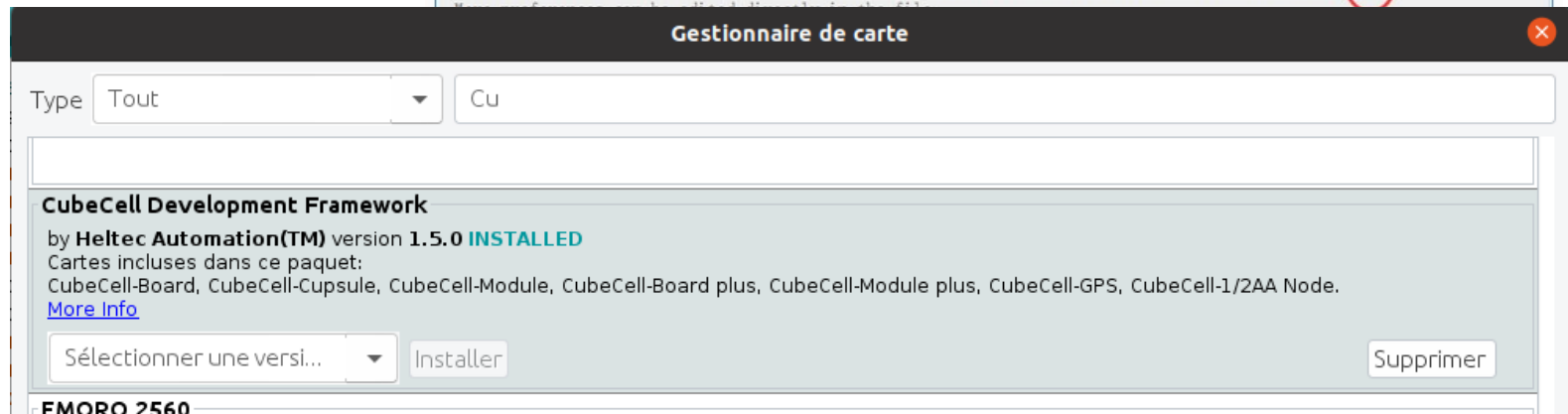
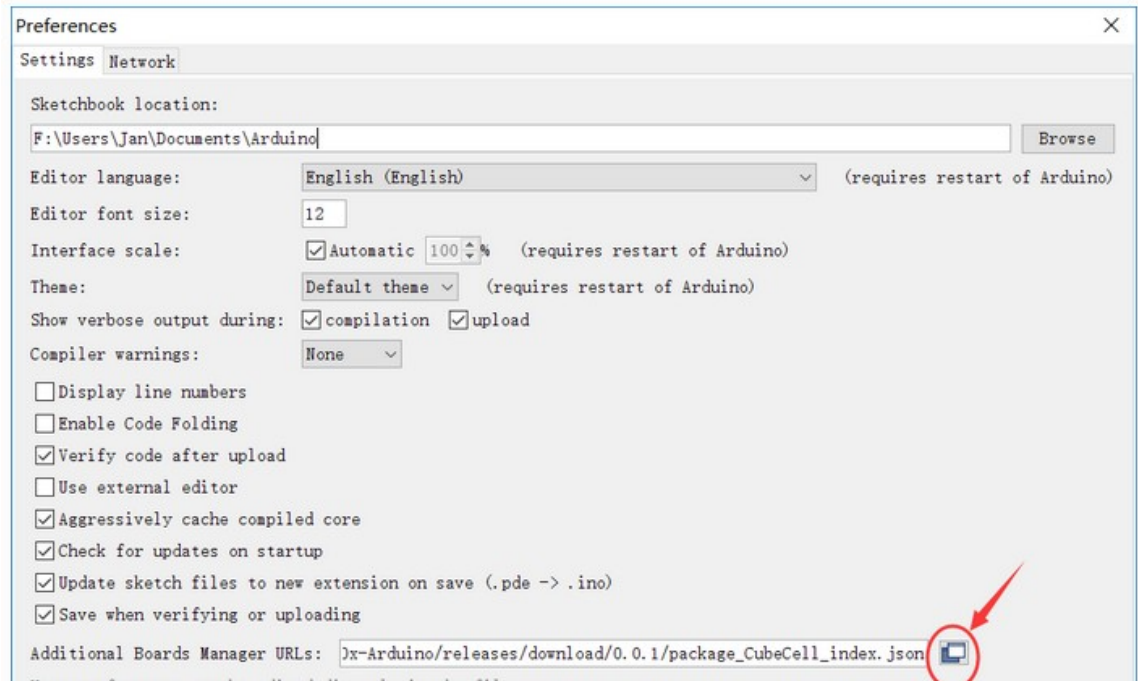
After the installation of the Arduino IDE we have to add the architectural platform to work with CubeCell boards.

This needs the addition of JSON link in the preferences of IDE:

https://github.com/HeITecAutomation/CubeCell-Arduino/releases/download/V1.5.0/package_CubeCell_index.json

CubeCell tools : (crossscompiler,loader)

After the preparation of the link to the CubeCell tools and libraries we can download them via the Board Manager as follows.





CubeCell : headers & events

The Arduino C/C++ code is compiled (cross-compiled) and loaded into flash memory of the board.

The initial section includes a number of libraries required to run the code for the SoM and the associated sensors.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <Wire.h>
#include "Adafruit_SHT4x.h"
Adafruit_SHT4x sht4 = Adafruit_SHT4x();
#include "Max44009.h"
Max44009 myLux(0x4A);

#define RX_TIMEOUT_VALUE 1000
static RadioEvents_t RadioEvents;
float txNumber;
bool lora_idle=true;
```

LoRaWan_APP.h is a complete library with the functions to operate with the basic LoRa transmissions as well as the LoRaWAN protocol.

As we use two sensors: SHT41 and Max44009 we need the corresponding libraries.

The operational timing of the LoRa modem (SX1262) is controlled via **RadioEvents** with timeout value.



CubeCell: low_power stage

The terminal is identified by the channel number associated to the ThingSpeak server. To write/read the data into/from this channel we need to provide read/write keys of 16-byte length.

Next code section defines the timer operations to put into sleep and to wakeup the processor.

```
unsigned long myChannelNumber =1626377; // put here your channel number
const char *myWriteAPIKey="3IN09682SQX3PT4Z" ;
const char *myReadAPIKey="9JVTP8ZHVTB9G4TT" ;
TimerEvent_t sleepTimer;
bool sleepTimerExpired; //Records whether our sleep/low power timer expired

static void wakeUp()
{
    sleepTimerExpired=true;
}

static void lowPowerSleep(uint32_t sleeptime) // corresponds to low_power stage
{
    sleepTimerExpired=false;
    TimerInit( &sleepTimer, &wakeUp );
    TimerSetValue( &sleepTimer, sleeptime );
    TimerStart( &sleepTimer );
    while (!sleepTimerExpired) lowPowerHandler();
    TimerStop( &sleepTimer );
}
```




CubeCell: LoRa parameters

LoRa modem radio parameters are registered in the structure **par**; . We have here the essential LoRa modulation parameters such as:

base frequency – **freq**,
spreading factor – **sf**,
signal modulation bandwidth – **bw**,
and code rate - **cr**.

```
typedef union
{
    uint8_t buff[28];
    struct
    {
        uint32_t  freq;           // GW - link frequency
        uint8_t   power;         // GW - emission power
        uint8_t   sf;           // GW - link spreading factor
        uint32_t  bw;           // GW - link signal bandwidth: [125E3, 250E3, 500E3]
        uint8_t   cr;           // GW - link coding rate: [5,6,7,8] in function
        uint8_t   aeskey[16];   // GW - AES key
        uint8_t   pad;
    } par;
} rt_lora_t;           // type definition
rt_lora_t rtlora;     // declaration
```



CubeCell: LoRa packet

LoRa packet sent to the receiver/gateway is structured as follows. Its format corresponds to the **μPython** structure declared as:

```
chan, wkey, temp, humi, lumi, count=struct.unpack('i16s4f', data)
```

where: `i` → `uint32_t`, `16s` → `char[16]`, `4f` → `float[4]`

```
typedef union
{
    uint8_t frame[36];
    struct
    {
        uint32_t    channel;           // channel number
        char        wkey[16];         // TS write key
        float       sens[4];          // temp, humi, lumi, count
    } pay;
} pack_t;
pack_t sdp; // packet to send
```



CubeCell: reading sensors – sensing phase

The following declarations allows us to keep some values alive during the deep-sleep stage. They are declared as **static**.

```
static uint16_t counter=0;
static float stemp=0.0, shumi=0.0, slumi=0.0;

float temp=0.0, humi=0.0, lumi=0.0;
```

There are two functions to read separately the values of temperature-humidity and luminosity sensors.

```
void readSHT41(float *tem, float *hum){ .. }
and
void readMAX44009(float *lux) { .. }
```

Note that both functions take the pointers to the variables used to return the sensor values.



CubeCell: setup () function

setup () section contains all necessary initialization to run the sender code. First we prepare the LoRa radio parameters via **setLoRaPar ()** function. Then we prepare the events to capture the end of data transmission without (**OnTxDone**) or with an error (**OnTxTimeout**). The prepared LoRa radio parameters are activated via **SetChannel ()** and **SetTxConfig ()** methods.

```
void setup() {  
    Serial.begin(9600); delay(300);  
    setLoRaPar(); // read lora parameters and TS  
parameters  
..  
    txNumber=0;  
    RadioEvents.TxDone = OnTxDone;  
    RadioEvents.TxTimeout = OnTxTimeout;  
    Radio.Init( &RadioEvents );  
    Radio.SetChannel( rtlora.par.freq );  
    Radio.SetTxConfig( MODEM_LORA, rtlora.par.power, 0, rtlora.par.bw,  
        rtlora.par.sf, rtlora.par.cr, 8, false,  
        true, 0, 0, false, 3000 );  
}
```



CubeCell: loop () function

loop () section runs in **two stages** : the entry into **low_power stage** is activated by:
`lowPowerSleep (ttsleep) ;`

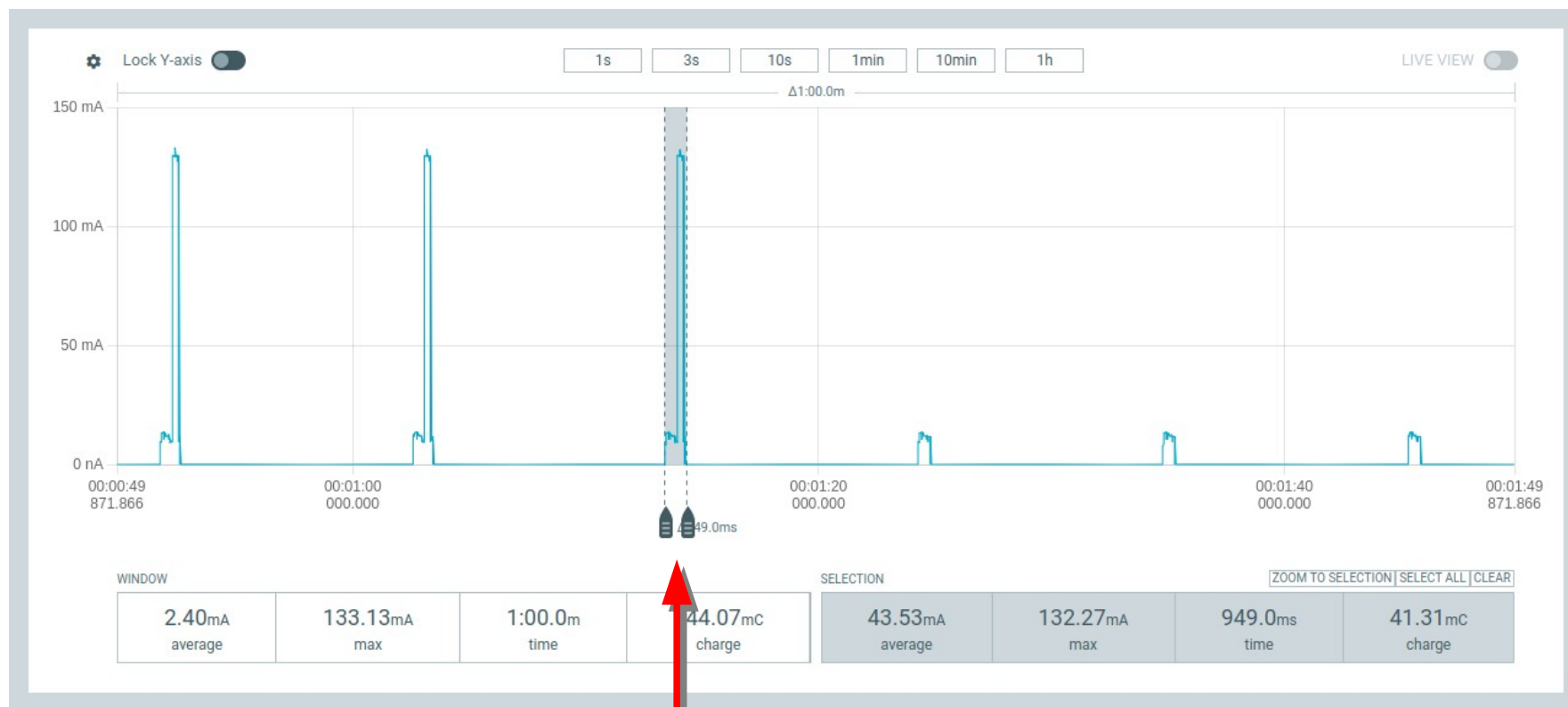
```
void loop()
{
  turnOffRGB(); counter++; digitalWrite(Vext, HIGH); delay(100);
  lowPowerSleep(ttsleep); // lora_idle = true;
  float ntemp, nhumi, nlumi, lux;
  readSHT41(&ntemp, &nhumi); readMAX44009(&nlumi);
  if(sent_valid(stemp, ntemp, 0.05))
  {
    if(lora_idle == true) // is true when TX is done !
    {
      stemp=ntemp; shumi=nhumi; slumi=nlumi;
      sdp.pay.channel=myChannelNumber;
      strncpy(sdp.pay.wkey, myWriteAPIKey, 16);
      sdp.pay.sens[0]=ntemp; sdp.pay.sens[1]=nhumi; sdp.pay.sens[2]=nlumi;
      Radio.Send( (uint8_t *)sdp.frame, 36 ); //send the package out
      lora_idle = false; delay(100);
      turnOnRGB(COLOR_SEND, 0); counter++; delay(100); digitalWrite(Vext, HIGH);
    }
  }
}
```

Power consumption : low_power stage



low_power stage – 31.82μA

Power consumption : full **high_power** stage



full **high_power** stage – 41.31mC

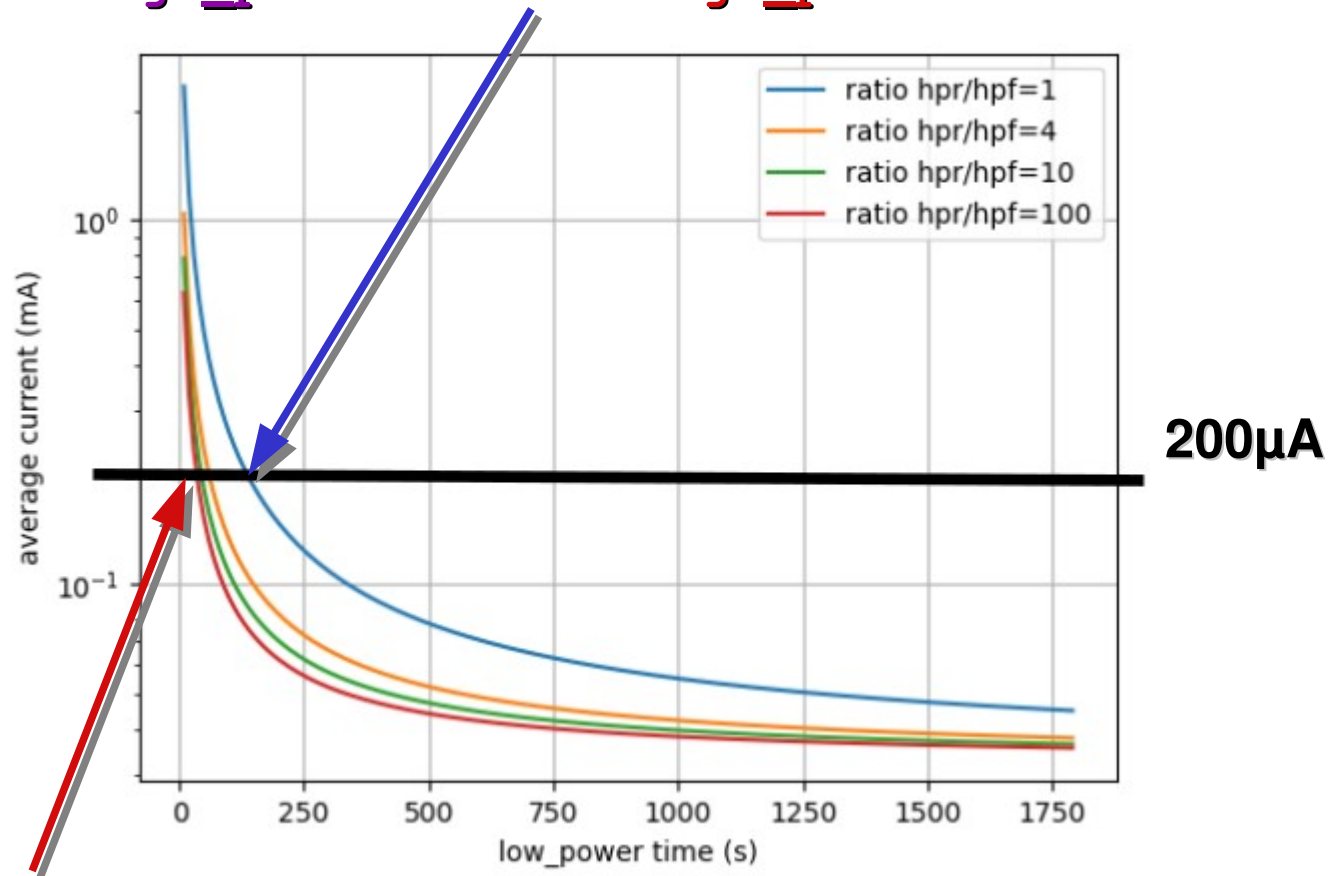
Power consumption : reduced **high_power** stage



reduced **high_power** stage – 5.94mC

Average current function of low_power stage time

high_power reduced/high_power full = 1 \Rightarrow 160s



high_power reduced/high_power full = 100 \Rightarrow 60s

Summary

IoT infrastructure & Devices

IoT devices identification:

@IP: port_n: channel_id

Low Power modes: stages & phases

Direct Terminals - WiFi (HT: RISC-V)

Power consumption on Direct Terminals

Remote Terminals – Lora (HT: RISC-V) , (CC: ARM-M0)

Power consumption on Remote Terminals

LoRa-WiFi Gateways & Complete IoT Architectures

