IoT Lab 2

Building Close Terminals and Gateways with MAC-WiFi (ESP-NOW)

In this lab we are going to study and experiment with ESP-NOW protocol that allows us to build simple yet very efficient communication links between the ESP32 based nodes.

2.1 Essential Features of ESP-NOW

ESP-NOW is a wireless communication protocol developed by Espressif for ESP32 SoCs. It operates at the MAC layer, providing fast and low-power communication between devices. Below are the key features:

1. **Peer-to-Peer Communication**
   - Devices can communicate directly without a router or access point.
   - ESP-NOW supports communication between multiple peers.
2. **Low Latency**
   - Operates at the MAC layer, bypassing the IP stack, which reduces communication latency significantly.
3. **Broadcast and Unicast Modes**
   - **Broadcast**: Send data to all paired devices.
   - **Unicast**: Target specific peers with their MAC addresses.
4. **Low Power**
   - ESP-NOW is designed for low-power applications, making it suitable for IoT devices.
5. **Encrypted Communication**
   - Supports secure communication using AES-128 encryption. Pre-shared keys (PSKs) are exchanged for encrypting data.
6. **Supports Up to 20 Peers**
   - An ESP32 can manage up to 20 peer devices.
7. **Platform Compatibility**
   - Works across ESP32, ESP8266, and other Espressif chips.
   - Use Cases of ESP-NOW
   - **IoT Networks**: Smart home devices, sensor nodes, and actuators.
   - **Data Logging**: Wireless data collection in a mesh or star topology.
   - **Wearable Devices**: Low-power communication for fitness trackers or medical devices.
   - **Remote Controls**: Reliable communication for remote control devices.

How to Use ESP-NOW on ESP32

1. **Initialization**
   - Enable the Wi-Fi interface in either STA or AP mode. STA mode is commonly used.
2. **Initialize ESP-NOW**
   - Use the `espnow` library in MicroPython (or equivalent API in Arduino or C).
   - Call `esp_now_init()` or similar to set up ESP-NOW.
3. **Pair Devices**
   - Add peers by specifying their MAC addresses.
   - You can also define whether the communication is encrypted.
4. **Send Data**
   - Use the `send()` method or its equivalent to transmit data to a specific peer or broadcast to all.
5. **Receive Data**
   - Set up a callback function to handle incoming data.
6. **Manage Peers**
   - Add, modify, or remove peers dynamically as needed.

Before programming ESP-NOW based application we need to know the MAC address of the receiver node. Try the following example.

```
-----------------------------------------------------------------------------------------------------
import network

# Initialize the network interface
wlan = network.WLAN(network.STA_IF)
# Activate the WLAN Interface
wlan.active(True)

# Check if the interface is active (connected)
if wlan.active():
    # Get the MAC address
    mac_address = wlan.config("mac")
    print(mac_address)
    print("Device MAC Address:", ":".join(["{:02X}".format(byte) for byte in mac_address]))
else:
    print("Wi-Fi is not active.")
-----------------------------------------------------------------------------------------------------

MPY: soft reboot
b'\x98=\xae\xb4\x00\x0c'
Device MAC Address: 98:3D:AE:B4:00:0C
-----------------------------------------------------------------------------------------------------
```

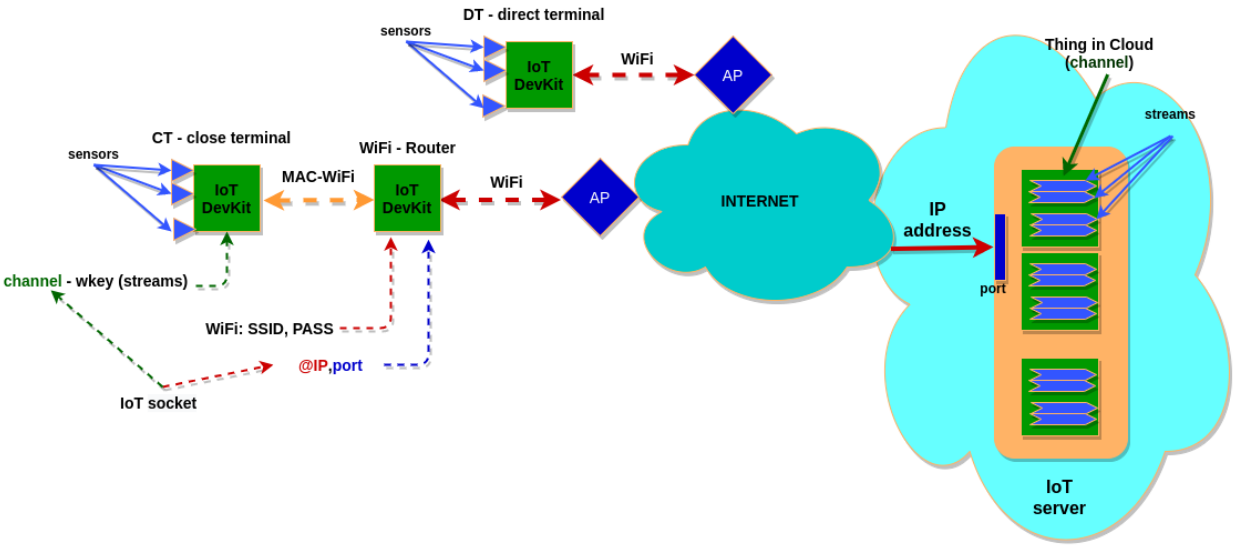

Fig 2.1 Sending sensor data from Close Terminal via WiFi-Router to ThingSpeak server. The IoT socket parameters decomposition: **channel** in **Close Terminal**, **@IP and port number** in **WiFi-Router**.

2.1 Simple sender and receiver nodes

The following are two programs to run on separate boards; one is sender and the other one is receiver.

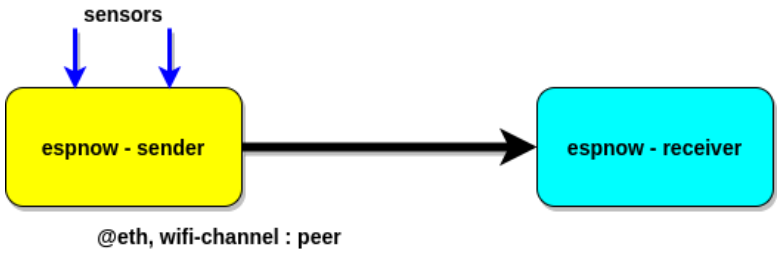

Fig. 2.1 Simple sender-receiver architecture to communicate with peer node (receiver)

2.1.1 ESP-NOW simple sender

The sender program uses integrated **button** on **DevKit** (`sig= Pin 0`) to provide simple messages : `start` and `stop`.

```
import network
from machine import Pin
import espnow
import utime
# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)  # Or network.AP_IF
sta.disconnect()
sta.active(True)
sta.config(txpower=5.0)
sta.config(channel=1)
sta.disconnect()
# Initialize ESP-NOW
esp = espnow.ESPNow()
esp.active(True)
print("now active")
#peer= b'\x48\xCA\x43\xD4\x05\x84'  # Replace with receiver's MAC address
peer= b'\xFF\xFF\xFF\xFF\xFF\xFF'  # Replace with broadcast MAC address
esp.add_peer(peer)
# Create a function to send data when a button is pressed (optional)
button_pin = Pin(0, Pin.IN, Pin.PULL_UP)
# Initialize variables for debouncing
last_button_state = 1  # Assuming the button is not pressed initially
debounce_delay = 50  # Adjust this value to your needs (milliseconds)
print(last_button_state)
while True:
    # Read the current state of the button
    current_button_state = button_pin.value()
    if current_button_state != last_button_state:
        # Wait for a short time to debounce the button
        utime.sleep_ms(debounce_delay)
        # Read the button state again to make sure it's stable
        current_button_state = button_pin.value()
        # If the button state is still different, it's a valid press
        if current_button_state != last_button_state:
            if current_button_state == 0:
                message = "start"
                print(f"Sending command : {message}")
                esp.send(peer, message)
            else:
                message = "stop"
                print(f"Sending command : {message}")
                esp.send(peer, message)

        last_button_state = current_button_state
```

```
MPY: soft reboot
now active
1
Sending command : start
Sending command : stop
Sending command : start
Sending command : stop
```

2.1.2 ESP-NOW simple receiver

```
import network
from machine import Pin
import espnow
# Initialize the network interface
wlan = network.WLAN(network.STA_IF)
# Activate the WLAN Interface
wlan.active(True)
if wlan.active():
    # Get the MAC address
    mac_address = wlan.config("mac")
    print(mac_address)
    print("Device MAC Address:", ":".join(["{:02X}".format(byte) for byte in mac_address]))
else:
    print("Wi-Fi is not active.")

import network

# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.config(txpower=5.0)
print("Running on channel:", sta.config("channel"))
sta.disconnect()   # Because ESP8266 auto-connects to last Access Point

e = espnow.ESPNow()
e.active(True)

while True:
    host, msg = e.recv()
    if msg:             # msg == None if timeout in recv()
        print(host, msg)
```

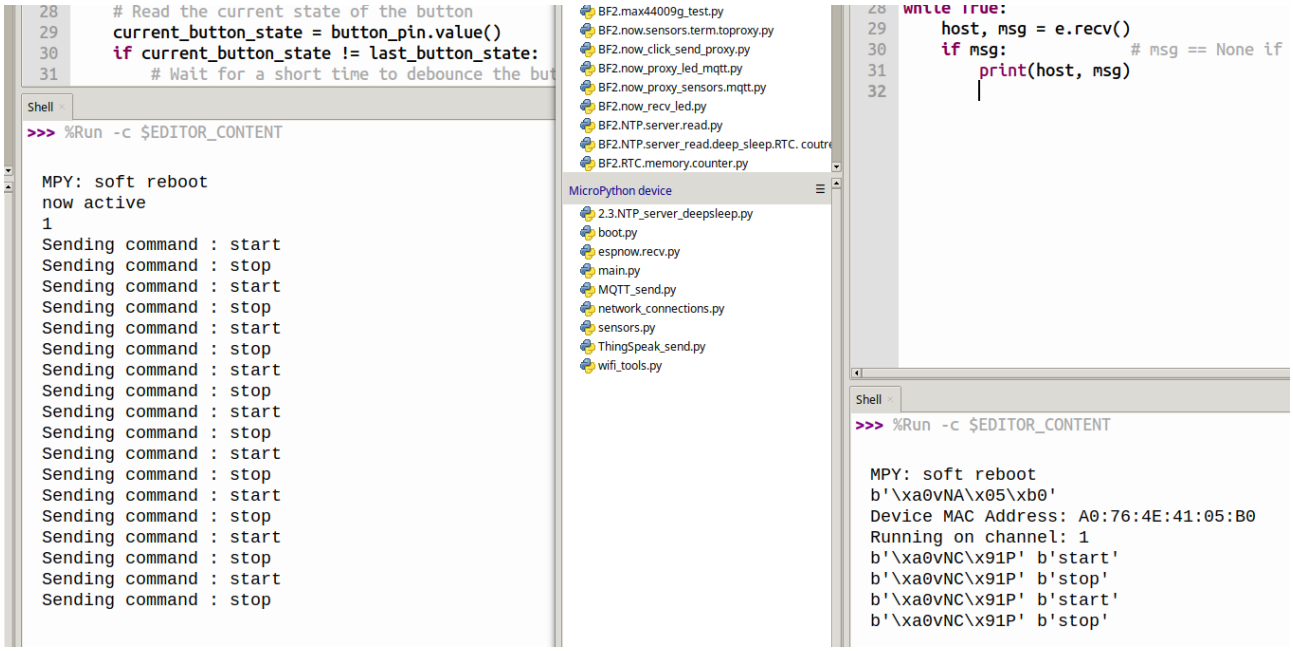

Fig. 2.2 ESPNOW **sender** (signal) and **receiver** terminals in Thonny IDE

To do:

Analyze the above examples. Note the use of the WiFi channel number:

`sta.config(channel=1)`

Modify the above programs adding sensor values formatted as data including:

`channel,wkey/topic,luminosity,temperature, humidity` **values:**

`data=ustruct.pack('i16s3f',1254,'smartcomputerlab',luminosity,temperature,humidity)`

2.1.3 ESP-NOW sender – sensor's values

In the following example we are going to send some actual values from sensors. These values as well as the additional information to be us**Fig. 3.4** ed lately to send the data to IoT server are structured as follows:

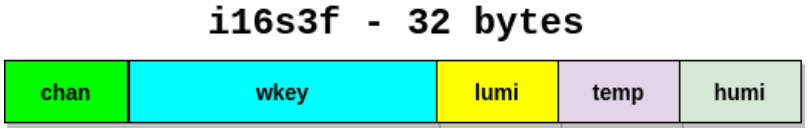

Fig. 2.3 Data packet structure including channel number **write key** or **topic** name and three sensor values: luminosity, temperature and humidity.

The sender side:

```
-----------------------------------------------------------------------------------------------
import network
from machine import Pin, deepsleep
import espnow
import utime, ustruct
from sensors import *
# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)  # Or network.AP_IF
#sta.disconnect()
sta.active(True)
sta.config(txpower=5.0)
sta.config(channel=1) # must be provide from gateway channel
sta.disconnect()      # For ESP8266
# Initialize ESP-NOW
esp = espnow.ESPNow()
esp.active(True)
print("now active")
#peer= b'\x54\x32\x04\x0B\x3C\xF8'  # Replace with receiver's MAC address
peer= b'\xFF\xFF\xFF\xFF\xFF\xFF'  #  broadcast MAC address
esp.add_peer(peer)
lumi,temp,humi=sensors(sda=8,scl=9)
print(lumi,temp,humi)
data=ustruct.pack('i16s3f',1254,'YOX31M0EDKO0JATK',lumi,temp,humi)
print(str(data))
esp.send(peer,data)
time.sleep(2)
deepsleep(6*1000)
-----------------------------------------------------------------------------------------------
```

The receiver side:

```
-----------------------------------------------------------------------------------------------
import network,espnow,ustruct
from machine import Pin
wlan = network.WLAN(network.STA_IF)
# Activate the WLAN Interface
wlan.active(True)
if wlan.active():
    # Get the MAC address
    mac_address = wlan.config("mac")
    print(mac_address)
    print("Device MAC Address:", ":".join(["{:02X}".format(byte) for byte in mac_address]))
else:
    print("Wi-Fi is not active.")

# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.config(txpower=5.0)
print("Running on channel:", sta.config("channel"))
sta.disconnect()   # Because ESP8266 auto-connects to last Access Point
e = espnow.ESPNow()
e.active(True)
while True:
    host, msg = e.recv()
    if msg:             # msg == None if timeout in recv()
        channel,wkey,lumi,temp,humi=ustruct.unpack('i16s3f',msg)
        print(host,channel,wkey,lumi,temp,humi)


-----------------------------------------------------------------------------------------------
```

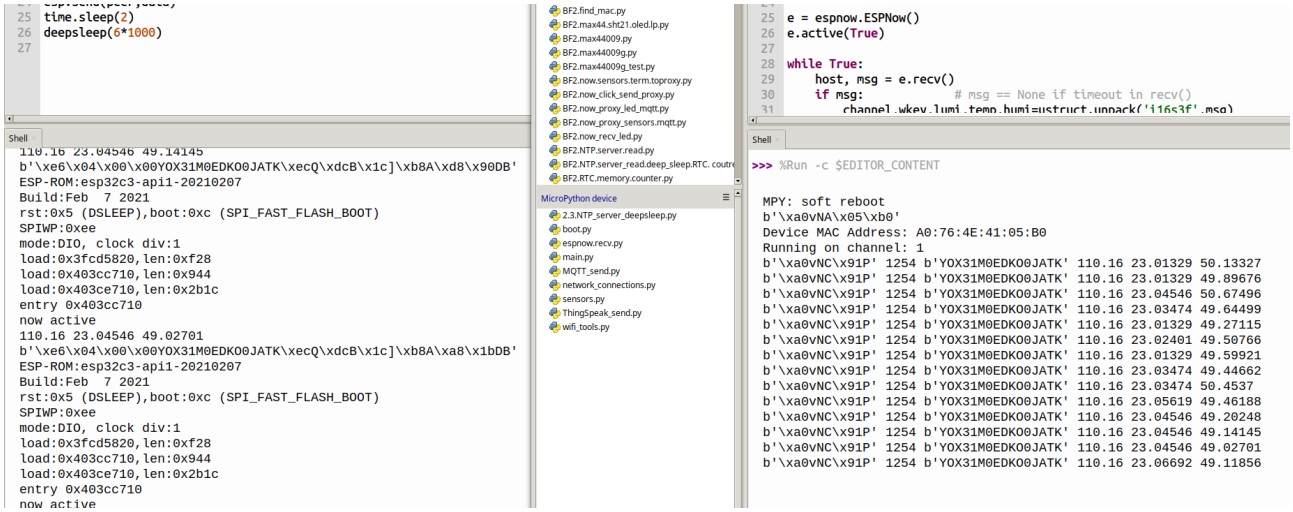

Fig. 2.4 ESPNOW **sender** (sensors) and **receiver** terminals in Thonny IDE

2.2 ESP-NOW sensor data receiver and gateway to ThingSpeak server

The **receiver node** can be extended to operate as a **gateway node** between **MAC-WiFi** and standard **Wi-Fi** communication links. In this configuration, the **sender node** must use the same **Wi-Fi channel (frequency)** that is employed for regular communication between the gateway and the **Wi-Fi access point**.

The gateway is programmed to **relay received packets** to the **MQTT server**, using the **topic value stored in the `wkey` field**. The **IP address** and **port number** of the MQTT server are known and configured at the gateway level.
This gateway functionality enables transparent forwarding of data from non-IP or MAC-level links to IP-based IoT services while preserving a simple and energy-efficient terminal design.

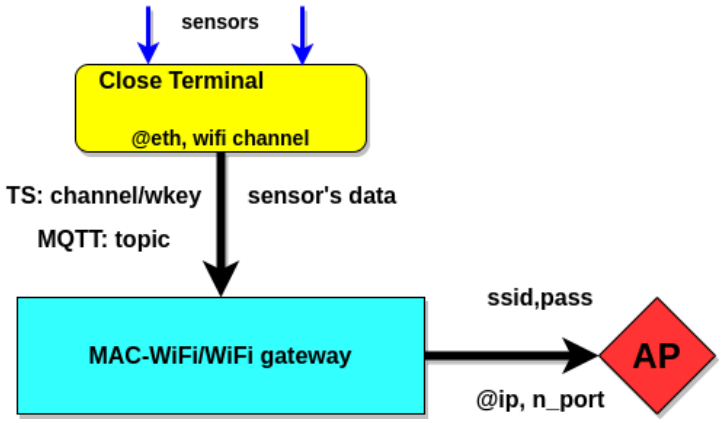

Fig. 2.5 Close Terminal operating on MAC-WiFi layer and corresponding (peer) gateway node

```
-------------------------------------------------------------------------------------------------------
import network, time, espnow, ustruct, urequests
from machine import Pin

def wifi_reset():   # Reset wifi to AP_IF off, STA_IF on and disconnected
  sta = network.WLAN(network.STA_IF); sta.active(False)
  ap = network.WLAN(network.AP_IF); ap.active(False)
  sta.active(True)
  sta.config(txpower=5.0)
  while not sta.active():
      time.sleep(0.1)
  sta.disconnect()   # For ESP8266
  while sta.isconnected():
      time.sleep(0.1)
  return sta, ap

# ThingSpeak API details
# THINGSPEAK_WRITE_API_KEY = 'YOX31M0EDKO0JATK' - this key is sent by the terminal
THINGSPEAK_URL = 'https://api.thingspeak.com/update'

# Function to send data to ThingSpeak
def send_data_to_thingspeak(wkey, lumi, temp, humi):
    try:
        url = f"{THINGSPEAK_URL}?api_key={wkey}&field1={lumi}&field2={temp}&field3={humi}"
        response = urequests.get(url)
        response.close()
        print("Data sent to ThingSpeak:", lumi, temp, humi)
    except Exception as e:
        print("Failed to send data:", e)

sta, ap = wifi_reset()  # Reset wifi to AP off, STA on and disconnected

sta.connect('Bbox-9ECEBF79', '54347A3EA6A1D6C36EF6A9E5156F7D')
while not sta.isconnected():  # Wait until connected...
    time.sleep(0.1)
sta.config(pm=sta.PM_NONE)  # ..then disable power saving
print(sta.ifconfig())

# Print the wifi channel used AFTER finished connecting to access point
print("Proxy running on channel:", sta.config("channel"))
e = espnow.ESPNow(); e.active(True)
for peer, msg in e:
```

```
        while True:
            host, data = e.recv()
            if data:
                chan, wkey, lumi, temp, humi = ustruct.unpack('i16s3f', data)   # wkey may be topic
                print(host)
                for_wkey="{:s}".format(wkey)
                print("wkey:"+str(for_wkey)+" lumi:"+str(lumi)+" temp:"+str(temp)+"
humi:"+str(humi))
                msg= "lumi:"+str(lumi)+"; temp:"+str(temp)+"; humi:"+str(humi)
                # Send data to ThingSpeak
                send_data_to_thingspeak(for_wkey,lumi, temp, humi)
```

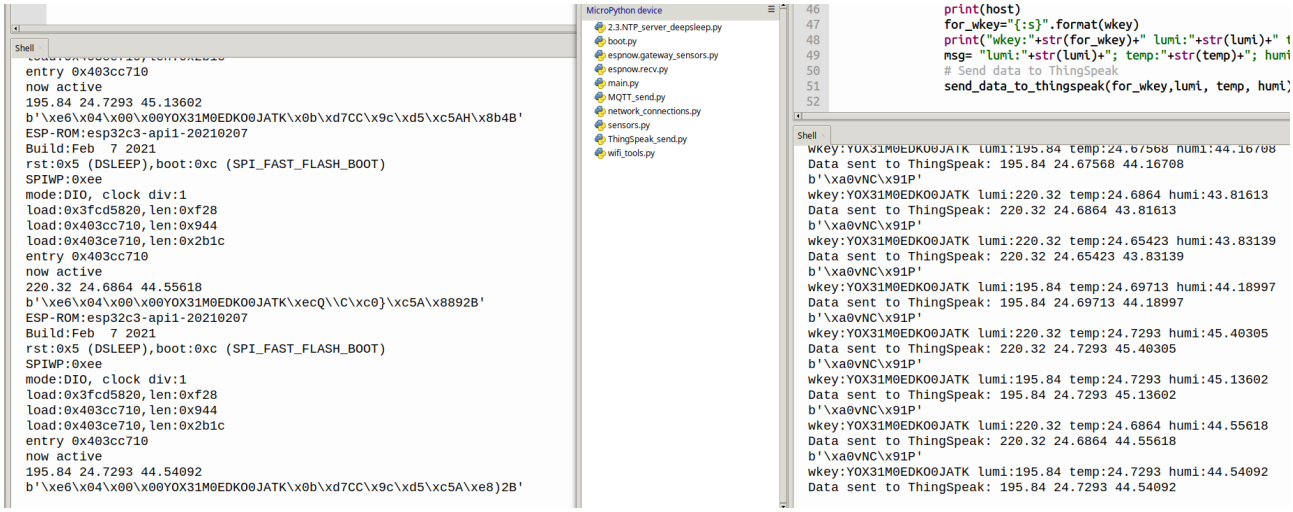

Fig. 2.6 ESPNOW **sender** (sensors) and **gateway** terminals in Thonny IDE

To do

Analyze and implement the same example with your credentials for WiFi and ThingSpeak server. Note that the terminal node sends the sensor's data and the write key (wkey) to the ThingSpeak channel.

Add OLED display to your gateway node.

2.3 ESP-NOW – close terminal (sender) with Low Power adaptive protocol

The following example introduces the same **adaptive mechanisms** previously presented for **direct (Wi-Fi) terminals**. These mechanisms include dynamic adjustment of the **cycle time** (low-power stage duration) and the **delta parameter**.

Note that the **delta condition** is intentionally separated from the **threshold condition**. This separation allows the system to respond more rapidly to **threshold-related events**, which often require immediate action. Initially, the **SoC operates at a minimum clock frequency of 20 MHz** to reduce power consumption. The clock frequency is increased to a **maximum of 160 MHz** only during the **transmission phase**, where higher processing and communication performance is required.

This dynamic frequency scaling further contributes to minimizing overall energy consumption while maintaining responsiveness when necessary.

2.3.1 Adaptive sender code

```
import network, esp
from machine import Pin, deepsleep, freq
import espnow
import utime, ustruct
from sensors import *
from rtc_tools import *
from nvs_tools import *
nvs_key="param"                             # key to NVS records
led = Pin(3, Pin.OUT)

def connect_send_espnow(chan,wkey,lumi,temp,humi):
    freq(160000000)         # maximum frequency
    sta = network.WLAN(network.STA_IF)  # Or network.AP_IF
    #sta.disconnect()
    sta.active(True)
    sta.config(txpower=5.0)
    sta.config(channel=1) # must be provide from gateway channel
    sta.disconnect()      # For ESP8266
    # Initialize ESP-NOW
    esp = espnow.ESPNow()
    esp.active(True)
    print("now active")
    #peer= b'\x54\x32\x04\x0B\x3C\xF8'  # Replace with receiver's MAC address
    peer= b'\xFF\xFF\xFF\xFF\xFF\xFF'  #  broadcast MAC address
    esp.add_peer(peer)

    data=ustruct.pack('i16s3f',chan,wkey,lumi,temp,humi)
    esp.send(peer,data)
    freq(20000000)

def main():
    freq(20000000)          # setting min frequency
    print("Reading ts from internal EEPROM...")
    len,ts_rparam = read_nvs_ts(nvs_key)
    if len:
        ts_chan,ts_wkey=ustruct.unpack("i16s",ts_rparam)
        print("len:",len,"ts_chan:",ts_chan,"ts_wkey:",ts_wkey.decode())
    len,pow_rparam = read_nvs_power(nvs_key)
    if len:
        cdef,cmax,dmin,dmax,tlow,thigh=ustruct.unpack("2i4f",pow_rparam)
        print("len:",len,", cdef:",cdef,", cmax:",cmax,", dmin:",dmin,", dmax:",dmax,",
tlow:",tlow,", thigh:",thigh)

    ncycle,npos,nneg= rtc_load_param()
    ssens= rtc_load_sensor(); sdelta= rtc_load_delta()
    print("ncycle:" +str(ncycle));
    lumi, temp, humi = sensors(sda=8, scl=9)
    print("Luminosity:", lumi, "lux"); print("Temperature:", temp, "C");print("Humidity:", humi,
"%")
    print("current: "+str(temp)+" saved: "+str(ssens));   # sensor is temperature
    print(dmin,dmax,sdelta);
    if abs(ssens-temp)>sdelta :  # testing delta and thresholds
        print("data to SEND")
        rtc_store_sensor(temp)
        led.on()
        if npos :
            if ncycle > 2:
                ncycle= int(ncycle/2)
            else:
                if sdelta< dmax:
                    sdelta = sdelta*2         # new delta
```

```
                    rtc_store_delta(sdelta)

        npos=npos+1; nneg=0  # positive and negative counters
        rtc_store_param(ncycle,npos,nneg)
        print(ts_wkey,lumi,temp,humi)
        connect_send_espnow(ts_chan,ts_wkey,lumi,temp,humi)
        print("data packet sent");led.off()

    elif temp>thigh or temp<tlow :
        ncycle=1; npos=0; nneg=0; rtc_store_param(ncycle,npos,nneg)
        sdelta = dmin; rtc_store(sdelta)
        print(ts_wkey,lumi,temp,humi)
        connect_send_espnow(ts_chan,ts_wkey,lumi,temp,humi)
        print("urgent data packet sent");led.off()

    else:
        print("data packet NOT sent")
        if nneg :
            if ncycle < cmax:
                  ncycle = int(ncycle*2)           # maximum factor 64
            else :
                if sdelta> dmin:
                    sdelta = sdelta/2
                    rtc_store_delta(sdelta)

        npos=0; nneg=nneg+1
        rtc_store_param(ncycle,npos,nneg)

    time.sleep(0.5)
    print(ncycle*cdef)
    print(sdelta)
    deepsleep(ncycle*cdef*1000)

main()
```

Fig 2.7 PPK2 10min diagram with **`low_power`** and **`high_power`** stages. Note low power (charge ~**43.69mC**) consumption for **`high_power`** stage with transmission due to the direct transmission of WiFi frames at MAC layer.