Internet of Things lab 4 Fall 2023

Group 2: Avhishek Biswas, Anuruddha Ekanayake, Amlan Balabantaray, Shaswati Behera

Task 01: Hello World

Requirements:

1. Finish the hello world example for each team member.

2. Record the procedure of setting up the link from your device to the IoT Hub with screenshots.

01. Development Plan:

a. Procedure of Solving the Problem

1. Install the ESP32 Arduino Core package (*https://dl.espressif.com/dl/package_esp32_index.json)*.

2. Download and install ESP-1ch-Gateway-v5.0-Azure.zip library to run LoRaWAN.

3. Configure the Gateway Sketch as follows:

ESP-sc-gway.h	ESP-sc-gway.ino	loramodem.h
Radio (frequency, SF, CAD)	Azure Connection String	Frequency channels
Hardware (OLED, Pinout, CFG_sx1276_radio)		
TTN		
WiFi (SSID)		

4. Upload the Hellow world message and Check NU-IoT WiFi Connection.

5. Setting up The Azure IoT Hub.

6. Create IoT device for the hub.

7. Test the gateway - cloud connection.

8. Setting up the end device by installing and configuring LMIC library.

9. Serial transmission using LMIC instead of RF95.

10. Streaming data into the cloud by configuring a resource on Azure Analytics Job.

b. Configuration Table

Requirement	Register Name	Register Function	Register Value
Clock Configuration	REG_GCLK_CLKCTRL	Configure Generic Clock Control	`GCLK_CLKCTRL_CLKEN
Timer Configuration	TC->CTRLA.reg	Timer Control A Register	`TC_CTRLA_MODE_COUNT16
Timer Frequency	TC->CC[0].reg	Timer Compare/Capture Register	compareValue
Timer Interrupt	TC->INTENSET.reg	Timer Interrupt Enable Set Register	TC_INTENSET.bit.MC0 = 1
NVIC Configuration	NVIC_EnableIRQ()	Nested Vector Interrupt Controller	TC3_IRQn

02. Development Process:

Connectivity and Hello World Example Output on IoT Hub

1. First connecting the Gateway to the Azure IoT Hub.
2. We get a connection success message.
3. Running LMIC example code and sending “Hello World”.
4. We will see the Gateway receiving and creating a message to transmit over NU-IOT wifi.
5. Finally, the query in the Azure website will be adding the messages.
6. We can view them by changing to a raw view that shows the JSON data.

Following are the screenshots of the process from each member:

Avhishek Biswas

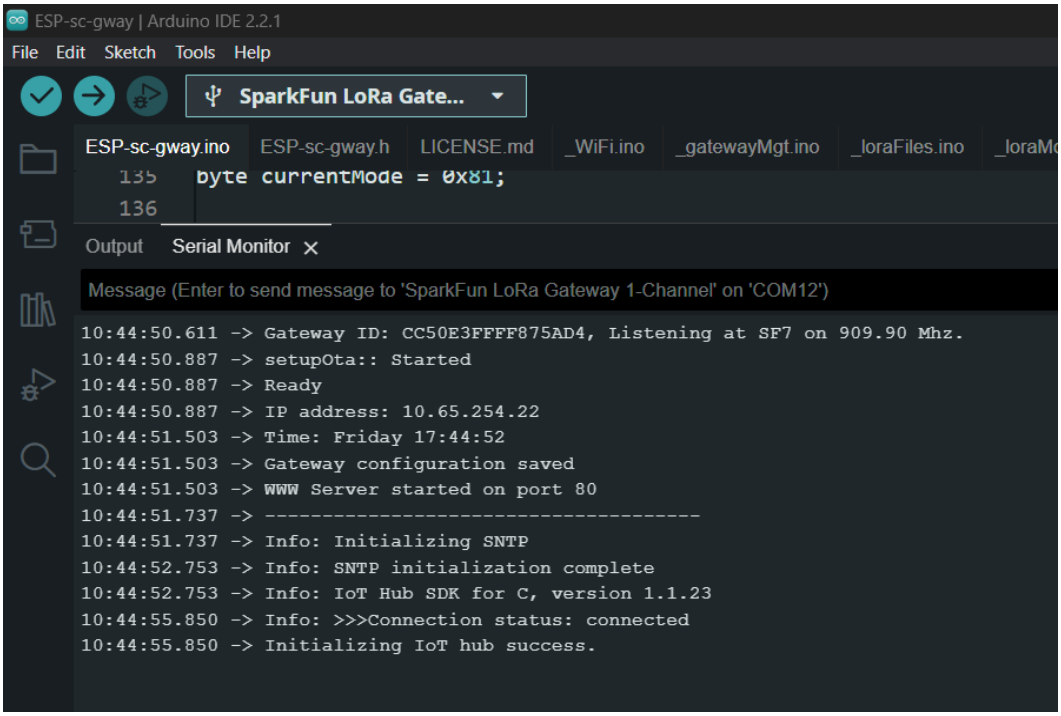


Figure 1: Gateway connection

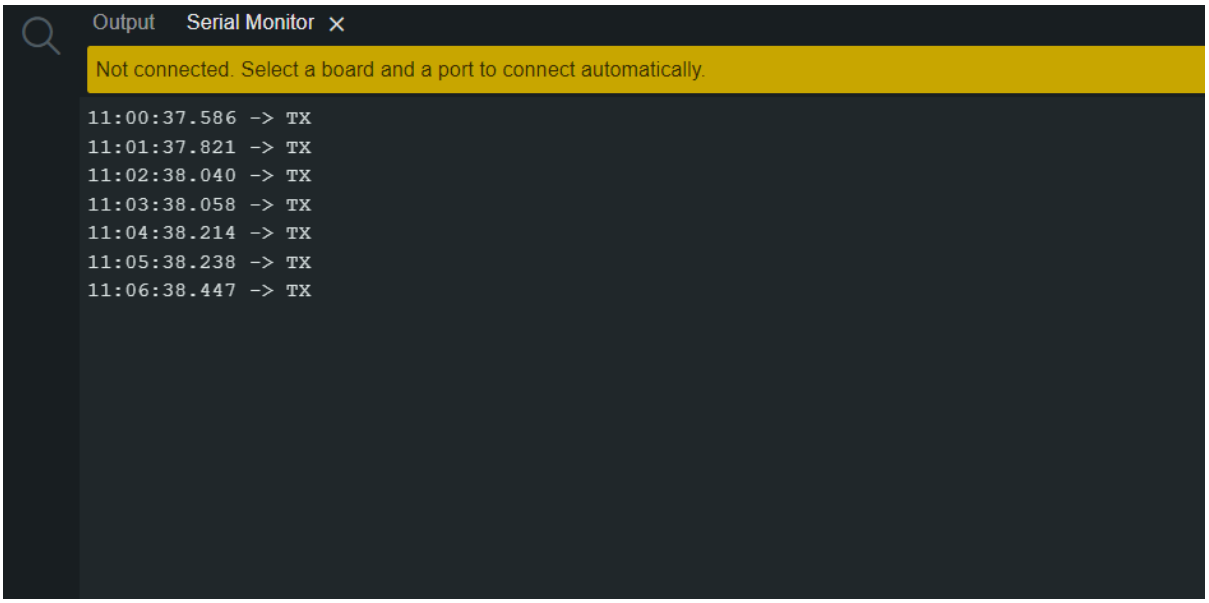


Figure 2: LMIC code sending hello message

Figure 3:Gateway reciving from LMIC Tx and sending to Azure

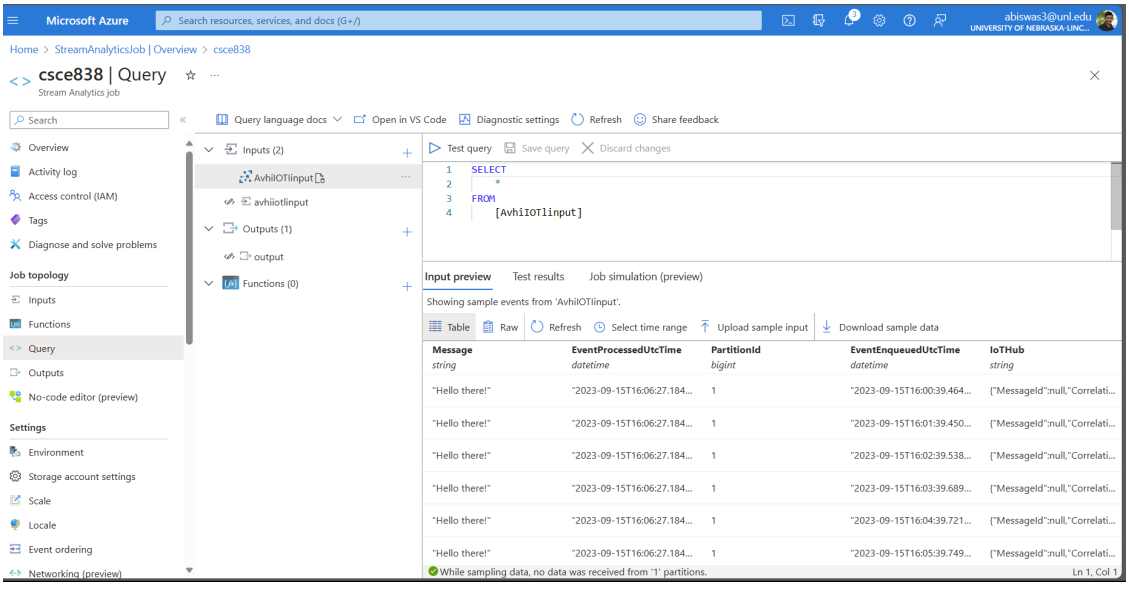
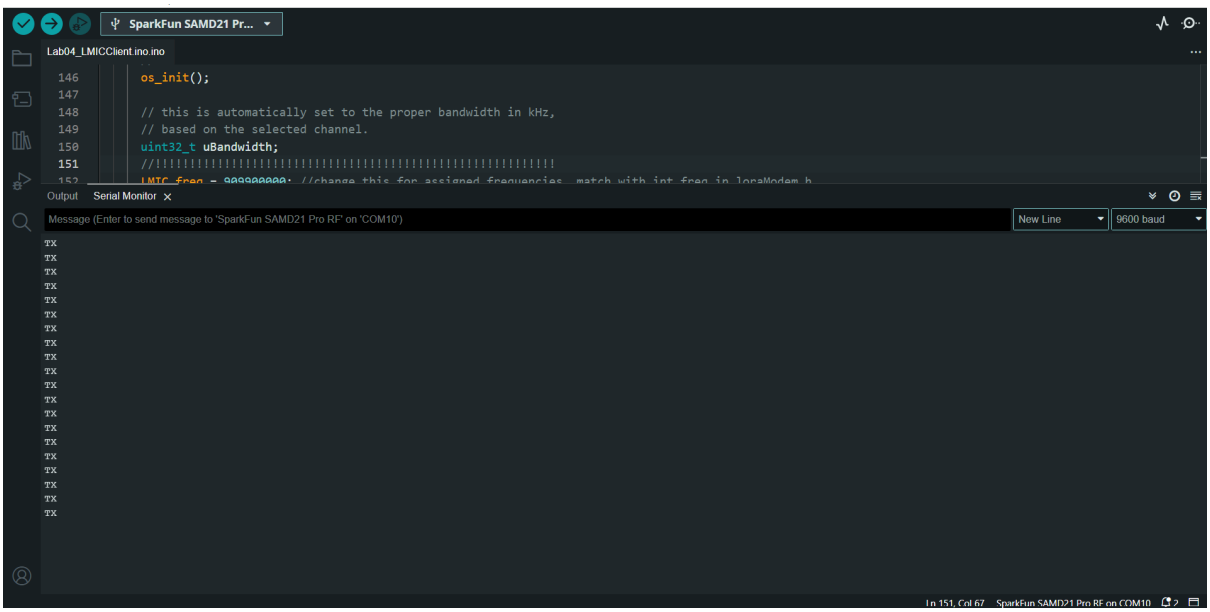
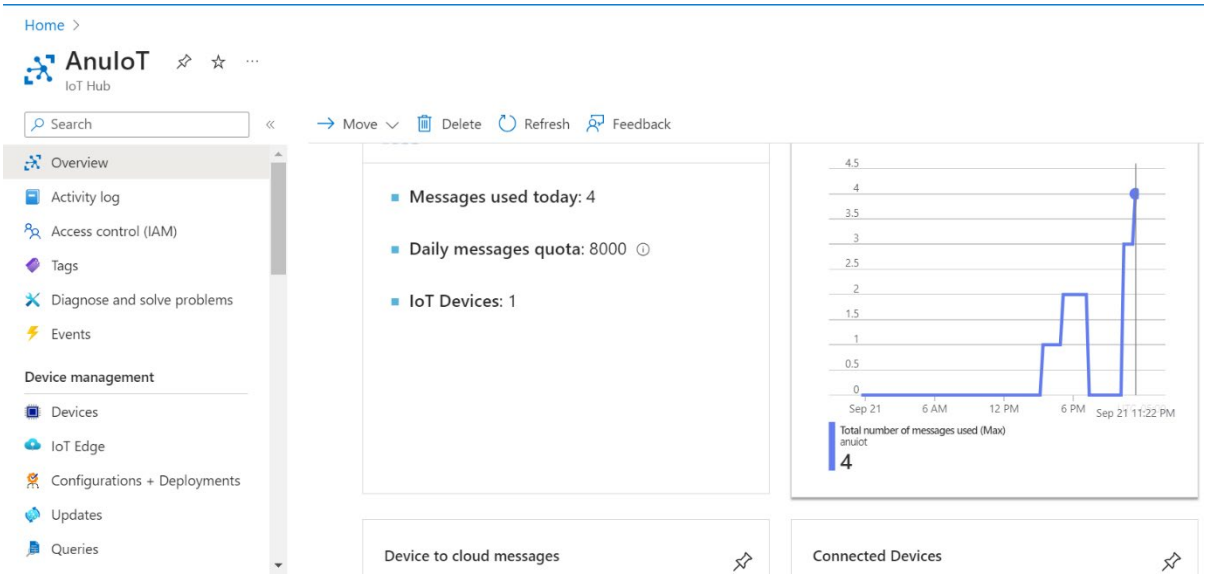
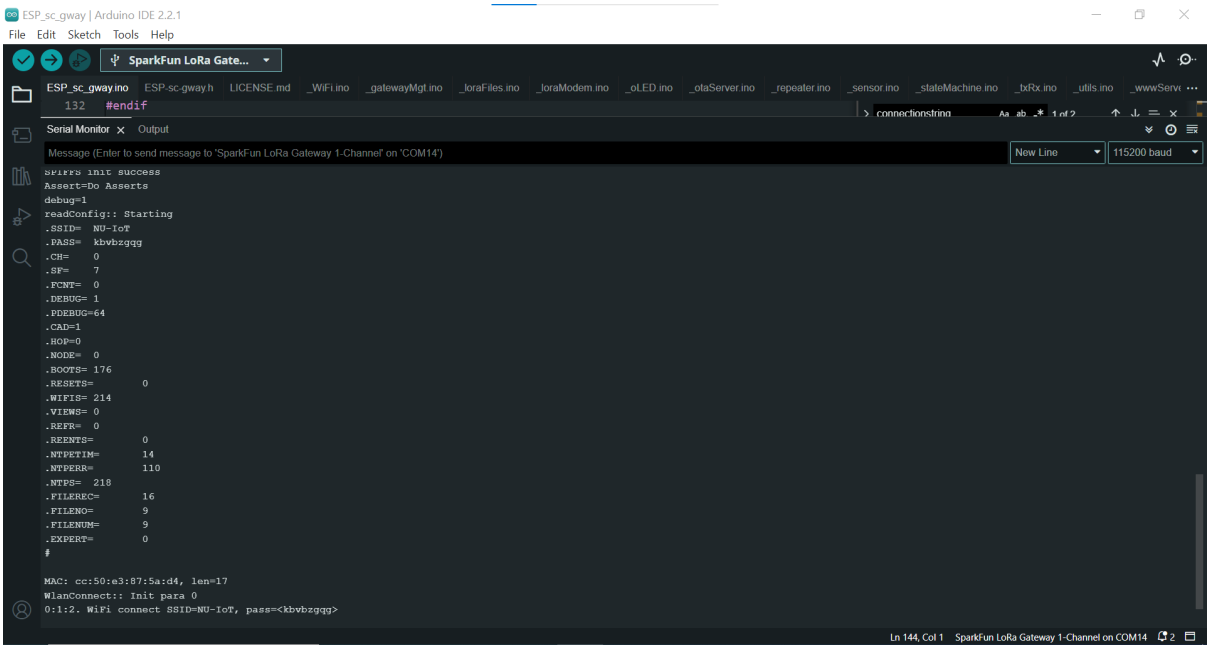


Figure 4: Azure query

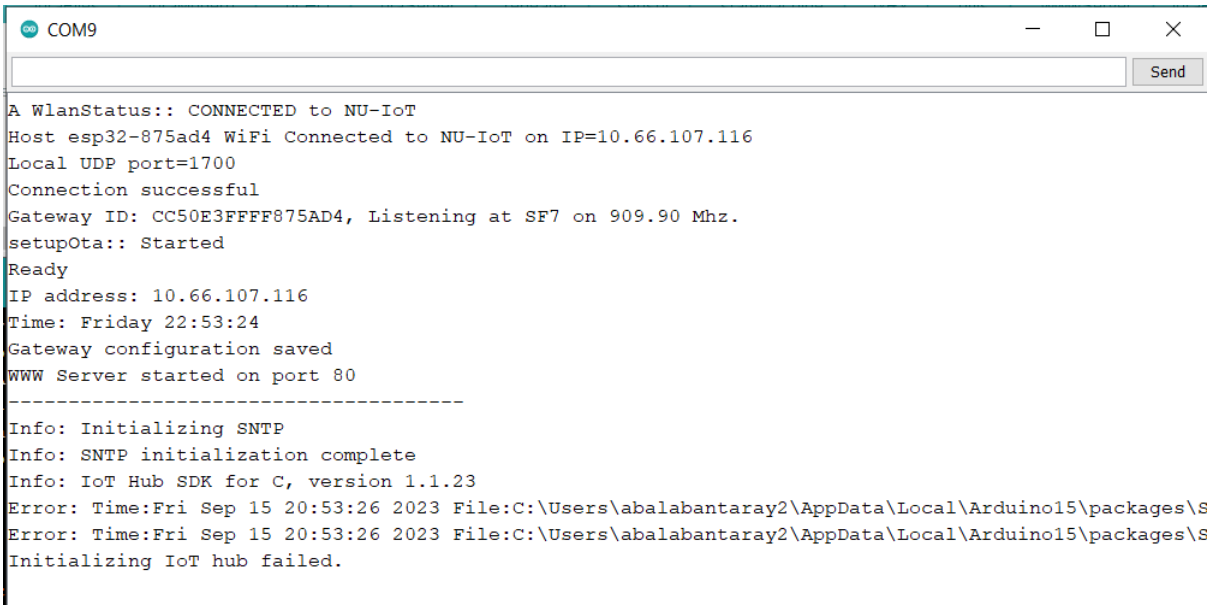
Anuruddha Ekanayake

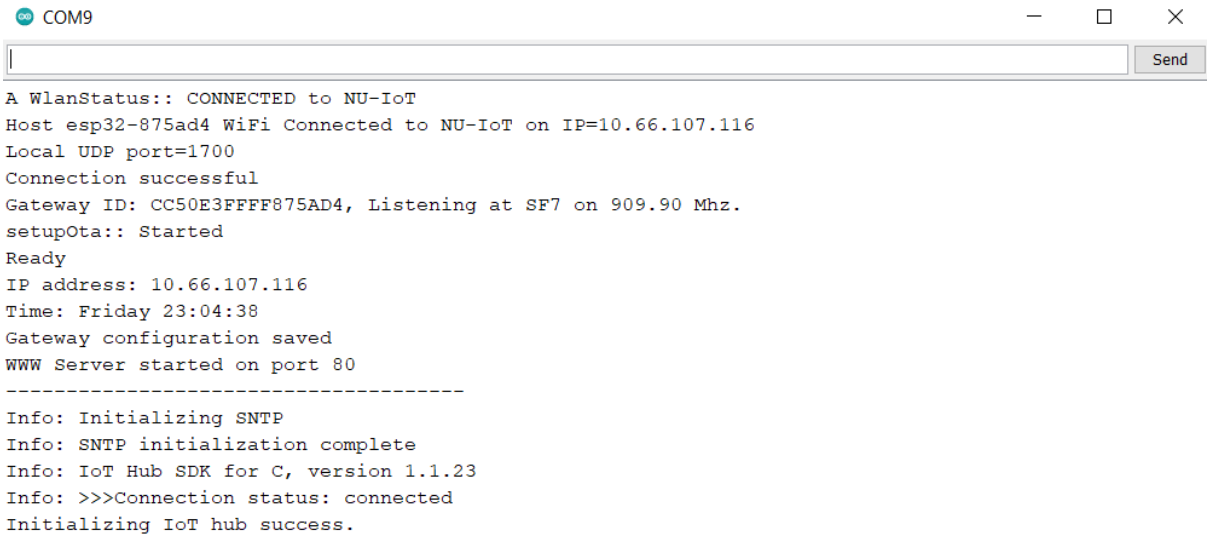




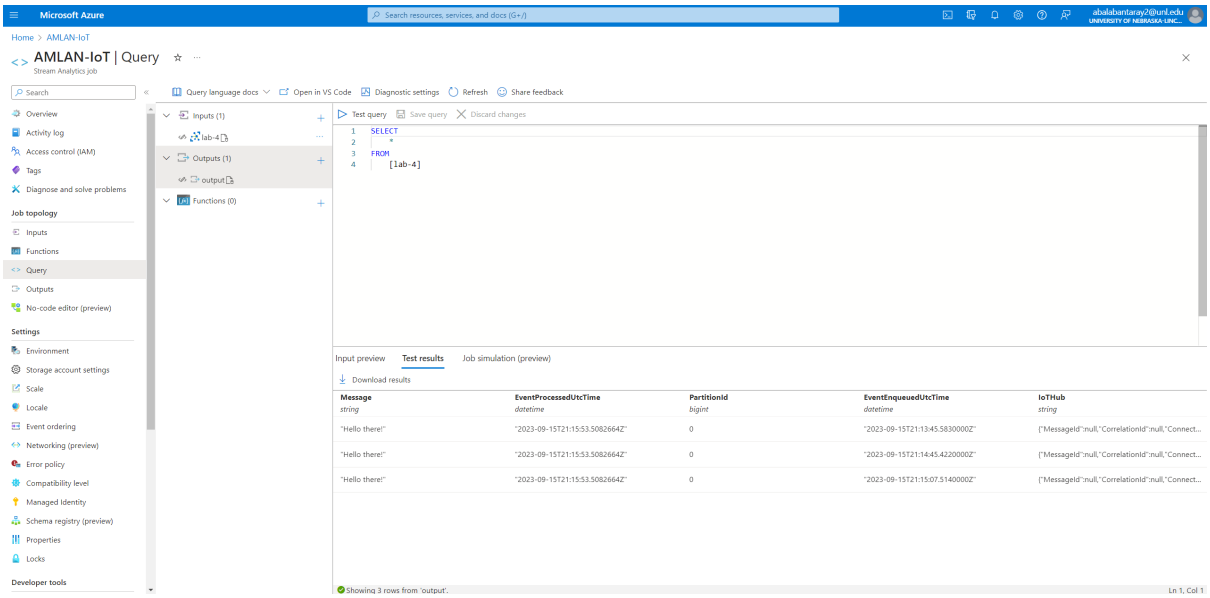


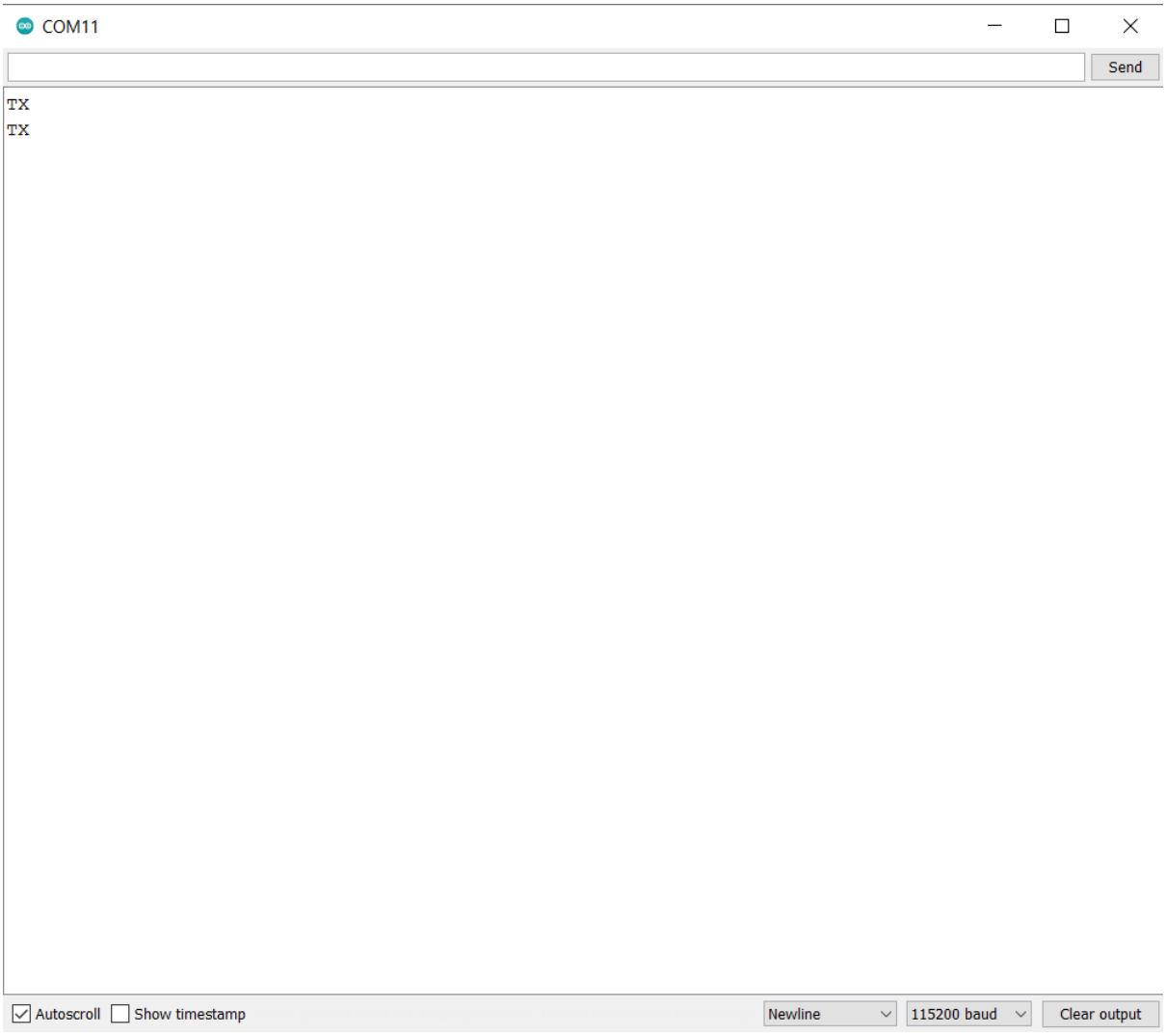
Amlan Balabantaray




COM9
Send
A WlanStatus:: CONNECTED to NU-IoT
Host esp32-875ad4 WiFi Connected to NU-IoT on IP=10.66.107.116
Local UDP port=1700
Connection successful
Gateway ID: CC50E3FFFF875AD4, Listening at SF7 on 909.90 Mhz.
setupOta:: Started
Ready
IP address: 10.66.107.116
Time: Friday 23:04:38
Gateway configuration saved
WWW Server started on port 80

Info: Initializing SNTP
Info: SNTP initialization complete
Info: IoT Hub SDK for C, version 1.1.23
Info: >>>Connection status: connected
Initializing IoT hub success.

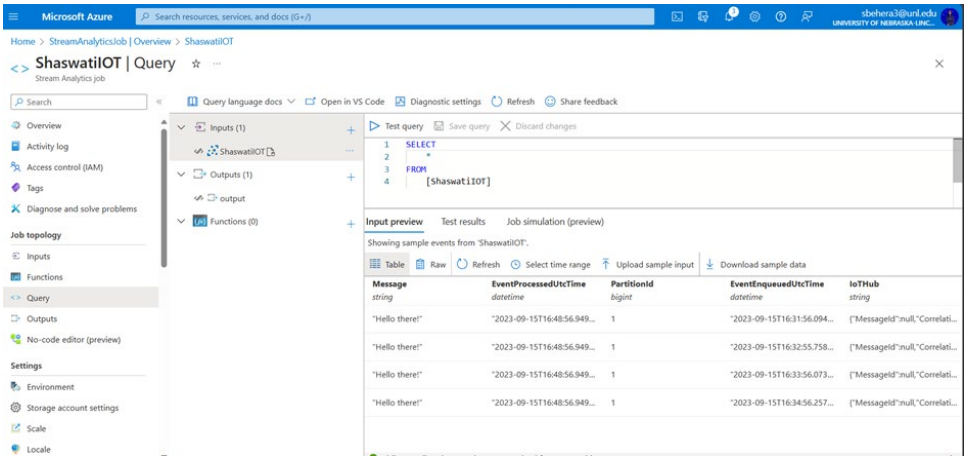

Microsoft Azure
Home > AMLAN-IoT
AMLAN-IoT | Query
Stream Analytics job
Query language docs
Open in VS Code
Diagnostic settings
Refresh
Share feedback
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Job topology
Inputs
Functions
Query
Outputs
No-code editor (preview)
Settings
Environment
Storage account settings
Scale
Locale
Event ordering
Networking (preview)
Error policy
Compatibility level
Managed Identity
Schema registry (preview)
Properties
Locks
Developer tools
Inputs (1)
lab-4
Outputs (1)
output
Functions (0)
Test query
Save query
Discard changes
SELECT
*
FROM
[lab-4]
Input preview
Test results
Job simulation (preview)
Download results
Message
EventProcessedUtcTime
PartitionId
EventEnqueuedUtcTime
IoTHub
"Hello there!"
"2023-09-15T21:15:53.5082664Z"
0
"2023-09-15T21:13:45.5830000Z"
"2023-09-15T21:14:45.4220000Z"
"2023-09-15T21:15:07.5140000Z"
Showing 3 rows from 'output'.

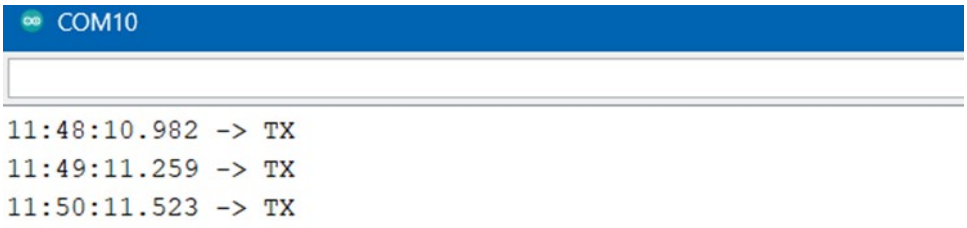


Shaswati Behera

```
11:52:23.893 -> MAC: cc:50:e3:87:5a:d4, len=17
11:52:23.893 -> WlanConnect:: Init para 0
11:52:24.036 -> 0:1:2. WiFi connect SSID=NU-IoT, pass=kbvbzgqg
11:52:33.074 -> A WlanStatus:: CONNECTED to NU-IoT
11:52:33.122 -> Host esp32-875ad4 WiFi Connected to NU-IoT on IP=10.65.254.22
11:52:33.312 -> Local UDP port=1700
11:52:33.312 -> Connection successful
11:52:33.979 -> Gateway ID: CC50E3FFFF875AD4, Listening at SF7 on 909.90 Mhz.
11:52:34.311 -> setupOta:: Started
11:52:34.311 -> Ready
11:52:34.311 -> IP address: 10.65.254.22
11:52:34.927 -> Time: Friday 18:52:30
11:52:34.927 -> Gateway configuration saved
11:52:34.927 -> WWW Server started on port 80
11:52:35.115 -> ------------------------------------
11:52:35.115 -> Info: Initializing SNTP
11:52:36.151 -> Info: SNTP initialization complete
11:52:36.151 -> Info: IoT Hub SDK for C, version 1.1.23
11:52:38.661 -> Info: >>>Connection status: connected
11:52:38.661 -> Initializing IoT hub success.
```

```
11:52:23.893 -> MAC: cc:50:e3:87:5a:d4, len=17
11:52:23.893 -> WlanConnect:: Init para 0
11:52:24.036 -> 0:1:2. WiFi connect SSID=NU-IoT, pass=kbvbzgqg
11:52:33.074 -> A WlanStatus:: CONNECTED to NU-IoT
11:52:33.122 -> Host esp32-875ad4 WiFi Connected to NU-IoT on IP=10.65.254.22
11:52:33.312 -> Local UDP port=1700
11:52:33.312 -> Connection successful
11:52:33.979 -> Gateway ID: CC50E3FFFF875AD4, Listening at SF7 on 909.90 Mhz.
11:52:34.311 -> setupOta:: Started
11:52:34.311 -> Ready
11:52:34.311 -> IP address: 10.65.254.22
11:52:34.927 -> Time: Friday 18:52:30
11:52:34.927 -> Gateway configuration saved
11:52:34.927 -> WWW Server started on port 80
11:52:35.115 -> ---------------------------------------
11:52:35.115 -> Info: Initializing SNTP
11:52:36.151 -> Info: SNTP initialization complete
11:52:36.151 -> Info: IoT Hub SDK for C, version 1.1.23
11:52:38.661 -> Info: >>>Connection status: connected
11:52:38.661 -> Initializing IoT hub success.
11:53:11.971 -> G addLog:: fileno=9, rec=17: 1 83 AB 0 CC 50 E3 FF FF 87 5A D4 {"rxpk":[{"tmst":49408600,"chan":0,"rfch":0,"freq":909.900024,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"
11:53:12.019 -> Hello there!
11:53:12.019 -> 48 65 6C 6C 6F 20 74 68 65 72 65 21
11:53:12.019 -> start sending events.
11:53:12.019 -> {"Message": "Hello there!"}
11:53:12.019 -> Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
11:53:12.729 -> Info: >>>Confirmation[0] received for message tracking id = 0 with result = IOTHUB_CLIENT_CONFIRMATION_OK
11:53:12.729 -> Sending data succeed
```





03. Test Plan:

1. IoT Hub : Verify Hello world message input on the Azure Stream Analytics Jobs resource input.

Component	Test Description	Results	Comment
Serial Output (Gateway)	Connecting to NU-IoT Wi-Fi	Connection successful	Successful
Serial Output (Gateway)	Testing gateway	Connected	Successful
Serial Output (Node)	LMICClient.ino uploading	Tx	Successful
Serial Output (Node)	Hello World Output	Hello World	Successful
IoT Hub Input	Streaming message		Successful

Task 02: Packet Transmission with LMIC

Requirements:

1. Use the code from the previous lab (Lab 3) and merge it with the LMIC example code for packet transmissions

2. Remove the radio operations in the Lab 3 code and instead, use the LMIC code.

3. Use your packet construction modules, average temperature reading modules, etc. from Lab 3.

01. Development Plan:

a. Procedure of Solving the Problem

1. First we needed to convert the communication process from RF95 to LMIC library, this is because they are both transmission drivers and cannot work in parallel.

2. So, we changed the Slave and Master code, such that:

    a. Slave 2 starts and counts till 5 secs to capture the temperature.

    b. At second = 2, it will transmit the packet.

    c. Slave 3 and Slave 4 will transmit similarly at their timing when the count turns 3 and 4 respectively.

    d. To avoid collision we have to carefully start the slaves,

    e. The Master listens for the 4 seconds and at the 5^{th} second it transmits to the gateway.

3. We changed the packet from a structure data type to a const char* variable so that LMIC can transmit it.

4. The key to receive is to uncomment the rx(rx_func) line in the Tx_done function, this activates the Rx and the code for master keeps on listening until it’s time to transmit.

b. Configuration Table

Requirement	Register Name	Register Function	Register Value
Clock Configuration	REG_GCLK_CLKCTRL	Configure Generic Clock Control	`GCLK_CLKCTRL_CLKEN
Timer Configuration	TC->CTRLA.reg	Timer Control A Register	`TC_CTRLA_MODE_COUNT16
Timer Frequency	TC->CC[0].reg	Timer Compare/Capture Register	compareValue
Timer Interrupt	TC->INTENSET.reg	Timer Interrupt Enable Set Register	TC_INTENSET.bit.MC0 = 1
NVIC Configuration	NVIC_EnableIRQ()	Nested Vector Interrupt Controller	TC3_IRQn

03. Test Plan:

1. IoT Hub : Verify Hello world message input on the Azure Stream Analytics Jobs resource input.

Component	Test Description	Results	Comment
Slave 2 transmitting	With a successful transmission we will see a serial output in Slave 2	Passed	Timing for slave is at 2 seconds.
Slave 3 transmitting	With a successful transmission we will see a serial output in Slave 3	Passed	Timing for slave is at 3 seconds.
Slave 4 transmitting	With a successful transmission we will see a serial output in Slave 4	Passed	Timing for slave is at 4 seconds.
Master receiving	We are able to receive and set the values of the temperature properly	Failed	There is sync error between the Master and the Slaves and sometimes it will not update the temperature variables.

Screenshots

```
03:03:11.713 -> Sending to Cloud
03:03:11.713 ->
03:03:11.713 -> TX
03:03:11.713 -> sent ...
03:03:12.207 -> Blue LED is on
03:03:12.795 -> RX
03:03:13.181 -> Blue LED is off
03:03:14.196 -> Blue LED is on
03:03:15.172 -> Blue LED is off
03:03:16.203 -> Blue LED is on
03:03:16.724 -> Sending to Cloud
03:03:16.724 ->
03:03:16.724 -> TX
03:03:16.724 -> sent ...
03:03:17.194 -> Blue LED is off
03:03:17.724 -> RX
03:03:18.201 -> Blue LED is on
03:03:19.186 -> Blue LED is off
03:03:20.202 -> Blue LED is on
03:03:21.201 -> Blue LED is off
03:03:21.704 -> Sending to Cloud
03:03:21.704 ->
03:03:21.704 -> TX
```

Figure 5: Master updating and sending packet to Gateway

```
00:48:18.746 -> Blue LED is off
00:48:19.761 -> Blue LED is on
00:48:20.760 -> Blue LED is off
00:48:21.752 -> Blue LED is on
00:48:21.806 -> Sending Master's Average Temperature.
00:48:21.806 -> SENDING TEMPERATURE OF NODES 2 : PACKET NUMBER :24
```

Figure 6 : Node 2 sending its packet

```
00:49:18.426 -> Blue LED is off
00:49:19.442 -> Blue LED is on
00:49:20.442 -> Blue LED is off
00:49:21.441 -> Blue LED is on
00:49:21.476 -> Sending Node 4's Average Temperature.
00:49:21.476 -> SENDING TEMPERATURE OFNODE 4 : PACKET NUMBER :166
```

Figure 7: Node 4 sending it's packet

Task 03: Sending Temperature Data to Azure Cloud

Requirements:

1. Maintain your packet structure from Lab 3 and make necessary changes in the gateway to prepare a JSON data packet.

2. Then, send packets with temperature sensor data to the Azure cloud every 60 seconds.

3. Instead of using timers, use LMIC's TX_INTERVAL.

01. Development Plan:

a. Procedure of Solving the Problem

1. As we are receiving the packets from the slaves, the master converts then to Packet variable which is a structure type.

2. We can get the nodeID from the packets, and then save the temperature for the nodes for that NodeID.

3. We then create a string in the transmission block to send the data to gateway.

4. we use the jsonTemperature.cstr() function to convert the String into the const char *

5. This will be captured by the gateway, as long as they are in the same frequency.

6. The gateway receives and prints this message.

03. Test Plan:

1. IoT Hub : Verify Hello world message input on the Azure Stream Analytics Jobs resource input.

Component	Test Description	Results	Comment
Serial Output	Output when the nodes temperature is received	Passed	Need to manage the start time of nodes to avoid collision
IoT Hub Input	Gateway reciving the code from the Master	Passed	Next will be viewing the JSON.

We kept the same packet structure but we converted the packets to CStrings and back using the following code:

```
char* PacketToCString(const Packet& packet) {

    // Allocate memory for the C-string. Size depends on your specific needs.

    char* cstr = new char[128];



    sprintf(cstr, "nodeID: %u, packetID: %u, timestamp: %u, payload: %f, error: %s, authID: %u",

            packet.nodeID, packet.packetID, packet.timestamp, packet.payload, packet.error, packet.authID);
```

```
    return cstr;
}

// returns packet from Cstring
Packet CStringToPacket(const char* cstr) {
    Packet packet;

    // Use sscanf to parse the string and populate the Packet fields
    int ret = sscanf(cstr, "nodeID: %hhu, packetID: %hu, timestamp: %u, payload: %f, error: %15s, authID: %hhu",
                     &packet.nodeID, &packet.packetID, &packet.timestamp, &packet.payload, packet.error, &packet.authID);
    // Optionally, you can check if all fields were successfully parsed
    if (ret != 6) {
        // Handle the error, e.g., by setting default values or logging an error message
    }
   return packet;
}
```

Screenshots

```
03:03:11.713 -> Sending to Cloud
03:03:11.713 ->
03:03:11.713 -> TX
03:03:11.713 -> sent ...
03:03:12.207 -> Blue LED is on
03:03:12.795 -> RX
03:03:13.181 -> Blue LED is off
03:03:14.196 -> Blue LED is on
03:03:15.172 -> Blue LED is off
03:03:16.203 -> Blue LED is on
03:03:16.724 -> Sending to Cloud
03:03:16.724 ->
03:03:16.724 -> TX
03:03:16.724 -> sent ...
03:03:17.194 -> Blue LED is off
03:03:17.724 -> RX
03:03:18.201 -> Blue LED is on
03:03:19.186 -> Blue LED is off
03:03:20.202 -> Blue LED is on
03:03:21.201 -> Blue LED is off
03:03:21.704 -> Sending to Cloud
03:03:21.704 ->
03:03:21.704 -> TX
```

Figure 8: Master reciving in the first 4 seconds and transmitting at the 5th

```
00:48:18.746 -> Blue LED is off
00:48:19.761 -> Blue LED is on
00:48:20.760 -> Blue LED is off
00:48:21.752 -> Blue LED is on
00:48:21.806 -> Sending Master's Average Temperature.
00:48:21.806 -> SENDING TEMPERATURE OF NODES 2 : PACKET NUMBER :24
```

```
00:49:18.426 -> Blue LED is off
00:49:19.442 -> Blue LED is on
00:49:20.442 -> Blue LED is off
00:49:21.441 -> Blue LED is on
00:49:21.476 -> Sending Node 4's Average Temperature.
00:49:21.476 -> SENDING TEMPERATURE OFNODE 4 : PACKET NUMBER :166
```

Figure 9 : Slave 2 and 4 transmitting at 2 different seconds

```
Sending data succeed
G addLog:: fileno=9, rec=36: 1 F3 FF 0 CC 50 E3 FF FF 87 5A D4 {"rxpk":[{"tmst":137967082,"chan
"Avhi": 31.06, "Amlan": 0.00, "Shaswati": 0.00, "Anu": 0.00}hasw
22 41 76 68 69 22 3A 20 33 31 2E 30 36 2C 20 22 41 6D 6C 61 6E 22 3A 20 30 2E 30 30 2C 20 22 53
start sending events.
{"Message": ""Avhi": 31.06, "Amlan": 0.00, "Shaswati": 0.00, "Anu": 0.00}hasw"}
Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
Info: >>>Confirmation[19] received for message tracking id = 19 with result = IOTHUB_CLIENT_CON
Sending data succeed
```

Figure 10 : Gateway reciving and transmitting the data

Task 04: Downloading the JSON file from Azure Cloud

Requirements:

1. Download the JSON file from Azure and share the contents of it in the report.

01. Development Plan:

a. Procedure of Solving the Problem

1. As the gateway is transmitting over the wifi,

2. We need to have the Azure Query tab open.

3. With time the Query will fill up,

4. Sometimes Refresh on the query tab helps in updating the list. This was a error we faced that the query was not updating even though the Gateway was transmitting.

Screenshots


```
},
{
    "Message": "Avhi: 30.91, Amlan: 0.00, Shaswati: 0.00, Anu: 0.00",
    "EventProcessedUtcTime": "2023-09-22T07:47:38.2394924Z",
    "PartitionId": 1,
    "EventEnqueuedUtcTime": "2023-09-22T06:59:28.4400000Z",
    "IoTHub": {
        "MessageId": null,
        "CorrelationId": null,
        "ConnectionDeviceId": "Avhi-IoT-1",
        "ConnectionDeviceGenerationId": "638303880806092683",
        "EnqueuedTime": "2023-09-22T06:59:28.4420000Z"
    }
},
{
    "Message": "Avhi: 30.96, Amlan: 0.00, Shaswati: 0.00, Anu: 0.00",
    "EventProcessedUtcTime": "2023-09-22T07:47:38.2394924Z",
    "PartitionId": 1,
    "EventEnqueuedUtcTime": "2023-09-22T06:59:33.3160000Z",
    "IoTHub": {
        "MessageId": null,
        "CorrelationId": null,
        "ConnectionDeviceId": "Avhi-IoT-1",
        "ConnectionDeviceGenerationId": "638303880806092683",
        "EnqueuedTime": "2023-09-22T06:59:33.3170000Z"
    }
}
```

Appendix:

```
#define CPU_HZ 48000000
#define TIMER_PRESCALER_DIV 1024
#define WINDOW_SIZE 5  // Size of the sliding window

// #include <sstream>
// #include <string>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <FlashStorage.h>
#include <TemperatureZero.h>

FlashStorage(error_Amlan, int);                 // Reserve a portion of flash memory, remove/comment when
upload to the transmitter
FlashStorage(error_Shaswati, int);
FlashStorage(error_Anuruddha, int);
FlashStorage(error_Avhishek, int);
FlashStorage(error_receive, int);

struct Packet {
  uint8_t nodeID;
  uint16_t packetID;
  uint32_t timestamp;
  float payload;
  char error[16];
  uint8_t authID;

  // Constructor to initialize the values
  Packet() : nodeID(0), packetID(0), timestamp(0), payload(0.0), authID(0) {
    // Initialize the error array to zeros
    memset(error, '0', sizeof(error));
}
};


char* PacketToCString(const Packet& packet) {
    // Allocate memory for the C-string. Size depends on your specific needs.
    char* cstr = new char[128];

    sprintf(cstr, "nodeID: %u, packetID: %u, timestamp: %u, payload: %f, error: %s, authID: %u",
            packet.nodeID, packet.packetID, packet.timestamp, packet.payload, packet.error, packet.authID);

    return cstr;
```

```
}

// returns packet from Cstring
Packet CStringToPacket(const char* cstr) {
    Packet packet;

    // Use sscanf to parse the string and populate the Packet fields
    int ret = sscanf(cstr, "nodeID: %hhu, packetID: %hu, timestamp: %u, payload: %f, error: %15s, authID:
%hhu",
                     &packet.nodeID, &packet.packetID, &packet.timestamp, &packet.payload, packet.error,
&packet.authID);
    // Optionally, you can check if all fields were successfully parsed
    if (ret != 6) {
        // Handle the error, e.g., by setting default values or logging an error message
    }
    return packet;
}



volatile bool canReadTemp = false;

volatile bool Slave2 = false;
volatile bool Slave3 = false;
volatile bool Slave4 = false;
volatile bool Master = false;

float node1_temperature = 0;
float node2_temperature = 0;
float node3_temperature = 0;
float node4_temperature = 0;


float tempBuffer[WINDOW_SIZE];  // Circular buffer to hold temperature values
int tempIndex = 0;  // Index to keep track of the oldest temperature value
float tempSum = 0;  // Sum of temperatures in the buffer
int tempCount = 0;  // Count of temperatures added to the buffer
int reportCount = 0;  // Count of readings before reporting
float avgTemperature = 0;  // To hold the average temperature
bool newAvgAvailable = false;  // Flag to indicate a new average is available

String jsonTemperature ;

int previouse_packet_id = 0;
int current_packet_id = 0;
```

```
int LED = 13; //Status LED is on pin 13
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
uint16_t packetCounter = 0;  // Initialize packet counter


//Define functions used in the program
void startTimer(int frequencyHz);
void configureClockTC4();
void configureTimerTC4(int frequencyHz);                //
void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz);
void TC4_Handler();
float Readtemp();
void write_error(int Node_name, int ecode);
void print_errors(char* err);


#define TX_INTERVAL 60000  //Delay between each message in millidecond.

// Pin mapping for SAMD21
const lmic_pinmap lmic_pins = {
    .nss = 12,//RFM Chip Select
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 7,//RFM Reset
    .dio = {6, 10, 11}, //RFM Interrupt, RFM LoRa pin, RFM LoRa pin
};


TemperatureZero TempZero = TemperatureZero();

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmoc/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

void onEvent (ev_t ev) {
}

osjob_t txjob;
osjob_t timeoutjob;
static void tx_func (osjob_t* job);
```

```
// Transmit the given string and call the given function afterwards
void tx(const char *str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1); // Wait a bit, without this os_radio below asserts, apparently because the state hasn't changed
yet
    LMIC.dataLen = 0;
    while (*str)
      LMIC.frame[LMIC.dataLen++] = *str++;
    LMIC.osjob.func = func;
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
    }

    // Enable rx mode and call func when a packet is received
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // RX _now_
    // Enable "continuous" RX (e.g. without a timeout, still stops after
    // receiving a packet)
    os_radio(RADIO_RXON);
    SerialUSB.println("RX");
    }

static void rxtimeout_func(osjob_t *job) {
    digitalWrite(LED_BUILTIN, LOW); // off
    }

static void rx_func (osjob_t* job) {
    // Blink once to confirm reception and then keep the led on
    digitalWrite(LED_BUILTIN, LOW); // off
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH); // on

    // Timeout RX (i.e. update led status) after 3 periods without RX
    os_setTimedCallback(&timeoutjob, os_getTime() + ms2osticks(3*TX_INTERVAL), rxtimeout_func);

    // Reschedule TX so that it should not collide with the other side's
    // next TX
    os_setTimedCallback(&txjob, os_getTime() + ms2osticks(TX_INTERVAL/2), tx_func);

    SerialUSB.print("Got ");
    SerialUSB.print(LMIC.dataLen);
    SerialUSB.println(" bytes");
    SerialUSB.write(LMIC.frame, LMIC.dataLen);
```

```
    SerialUSB.println();

    // Restart RX
    rx(rx_func);
    }

static void txdone_func (osjob_t* job) {
    delay(1000);
    rx(rx_func);
    }

    // log text to USART and toggle LED
static void tx_func (osjob_t* job) {
    // say hello
    float temperature = TempZero.readInternalTemperature();
    String temperatureString = String(temperature, 2);
    String AvhiTemperature = "Avhi Temp :" + temperatureString;
    // tx("First message going in...!", txdone_func);
    // tx(AvhiTemperature.c_str(), txdone_func);
    // reschedule job every TX_INTERVAL (plus a bit of random to prevent
    // systematic collisions), unless packets are received, then rx_func
    // will reschedule at half this time.
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), tx_func);
    }

    // application entry point
void setup()
{
    SerialUSB.begin(115200);
    // while(!SerialUSB);
    SerialUSB.println("Starting");
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) |
    GCLK_GENCTRL_GENEN |
    GCLK_GENCTRL_SRC_OSCULP32K |
    GCLK_GENCTRL_DIVSEL;
    while(GCLK->STATUS.bit.SYNCBUSY);                              // wait while synchronizing the GCLK-
>STATUS.bit

                                                                    // WDT clock = clock gen 2
    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT |
    GCLK_CLKCTRL_CLKEN |
    GCLK_CLKCTRL_GEN_GCLK2;
    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0x8; // Set period for chip reset from the datasheet
```

```
WDT->INTENSET.bit.EW = 1; // Enable early warning interrupt
WDT->CTRL.bit.WEN = 0;
SerialUSB.begin(115200);
TempZero.init();


pinMode(LED_BUILTIN, OUTPUT);

// initialize runtime env
os_init();

// this is automatically set to the proper bandwidth in kHz,
// based on the selected channel.
uint32_t uBandwidth;
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LMIC.freq = 909900000; //change this for assigned frequencies, match with int freq in loraModem.h
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
uBandwidth = 125;
LMIC.datarate = US915_DR_SF7;         // DR4
LMIC.txpow = 21;

int previouse_packet_id = 0;
int current_packet_id = 0;



int LED = 13; //Status LED is on pin 13
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received

float frequency = 910; //Broadcast frequency
uint16_t packetCounter = 0;  // Initialize packet counter



// disable RX IQ inversion
LMIC.noRXIQinversion = true;

// This sets CR 4/5, BW125 (except for EU/AS923 DR_SF7B, which uses BW250)
LMIC.rps = updr2rps(LMIC.datarate);

SerialUSB.print("Frequency: "); SerialUSB.print(LMIC.freq / 1000000);
          SerialUSB.print("."); SerialUSB.print((LMIC.freq / 100000) % 10);
          SerialUSB.print("MHz");
SerialUSB.print("  LMIC.datarate: "); SerialUSB.print(LMIC.datarate);
SerialUSB.print("  LMIC.txpow: "); SerialUSB.println(LMIC.txpow);
```

```
    // This sets CR 4/5, BW125 (except for DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    SerialUSB.println("Started");
    SerialUSB.flush();

    // setup initial job
    os_setCallback(&txjob, tx_func);

    WDT->CTRL.bit.ENABLE = 1;
    startTimer(1);

}


void loop()
{
    // execute scheduled jobs and events
    os_runloop_once();
    avgTemperature = Readtemp();
    node1_temperature = avgTemperature;
  // recieve from the slaves for the first 4 seconds
  if (reportCount < 4) //&& Slave2 == false
    {
      Packet receivedPacket = CStringToPacket((const char*)LMIC.frame);
      timeSinceLastPacket = millis(); //Timestamp this packet

      SerialUSB.println();
      SerialUSB.println();

      // Print the received packet details
      SerialUSB.println("Got message from Node ID: ");
      uint8_t nodeid = receivedPacket.nodeID;

      SerialUSB.print(nodeid);
      if(nodeid == 2){
        node2_temperature = receivedPacket.payload;
        SerialUSB.println(node2_temperature);}
      else if(nodeid == 3){
        node3_temperature = receivedPacket.payload;
        SerialUSB.println(node3_temperature);
```

```
      }
      else if(nodeid == 4){
        node3_temperature = receivedPacket.payload;
        SerialUSB.println(node3_temperature);
      }
    }
  if(reportCount == 4 && Master == false){
  // else if(reportCount == 4){
    delay(500);
    SerialUSB.println("Sending to Cloud");
    jsonTemperature += "\Avhi\: " + String(node1_temperature, 2) + ", ";
    jsonTemperature += "\Amlan\: " + String(node2_temperature, 2) + ", ";
    jsonTemperature += "\Shaswati\: " + String(node3_temperature, 2) + ", ";
    jsonTemperature += "\Anu\: " + String(node4_temperature, 2);

    SerialUSB.println(jsonTemperature);
    // jsonTemperature = "";
    tx(jsonTemperature.c_str(), txdone_func);
    SerialUSB.println("sent ...");
    Master = true;
    jsonTemperature = "";
  }
}

void startTimer(int frequencyHz) {
  configureClockTC4();
  configureTimerTC4(frequencyHz);
}
void configureClockTC4() {
  REG_GCLK_CLKCTRL = (uint16_t)(GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_ID_TC4_TC5);
  while (GCLK->STATUS.bit.SYNCBUSY == 1);
}

void configureTimerTC4(int frequencyHz) {
  TcCount16* TC = (TcCount16*)TC4;
  TC->CTRLA.reg &= ~TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  TC->CTRLA.reg |= TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  setTimerFrequencyTC4(TC, frequencyHz);
  TC->INTENSET.reg = 0;
  TC->INTENSET.bit.MC0 = 1;
```

```
  NVIC_EnableIRQ(TC4_IRQn);
  TC->CTRLA.reg |= TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz) {
  int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
  TC->CC[0].reg = compareValue;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void TC4_Handler() {
  static bool isLEDOn = false;
  TcCount16* TC = (TcCount16*)TC4;
  if (TC->INTFLAG.bit.MC0 == 1)
  {
    TC->INTFLAG.bit.MC0 = 1;
    WDT->CLEAR.reg = WDT_CLEAR_CLEAR_KEY;
    canReadTemp = true;
   digitalWrite(PIN_LED_13, isLEDOn);
   SerialUSB.println(isLEDOn ? "Blue LED is on" : "Blue LED is off");
   isLEDOn = !isLEDOn;
   Master = false;
  }
}

float Readtemp() {
  if (canReadTemp) {
    float temperature = TempZero.readInternalTemperature();

    // Remove the oldest temperature from the sum
    tempSum -= tempBuffer[tempIndex];

    // Add the new temperature to the buffer and sum
    tempBuffer[tempIndex] = temperature;
    tempSum += temperature;

    // Update the index for the oldest temperature
    tempIndex = (tempIndex + 1) % WINDOW_SIZE;

    // Update the count of temperatures added to the buffer
    if (tempCount < WINDOW_SIZE) {
      tempCount++;
```

```
    }

    // Increment the report counter
    reportCount++;

    // If 5 readings have been taken, calculate the average temperature
    if (reportCount >= WINDOW_SIZE) {
      avgTemperature = tempSum / tempCount;
      newAvgAvailable = true;  // Set the flag
      reportCount = 0;  // Reset the report counter
    }

    canReadTemp = false;

  }
  return avgTemperature;
}
```

Slave 2 :

```
#define CPU_HZ 48000000
#define TIMER_PRESCALER_DIV 1024
#define WINDOW_SIZE 5  // Size of the sliding window

// #include <sstream>
// #include <string>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <FlashStorage.h>
#include <TemperatureZero.h>

FlashStorage(error_Amlan, int);                 // Reserve a portion of flash memory, remove/comment when
upload to the transmitter
FlashStorage(error_Shaswati, int);
FlashStorage(error_Anuruddha, int);
FlashStorage(error_Avhishek, int);
FlashStorage(error_receive, int);

struct Packet {
  uint8_t nodeID;
  uint16_t packetID;
  uint32_t timestamp;
```

```
  float payload;
  char error[16];
  uint8_t authID;

  // Constructor to initialize the values
  Packet() : nodeID(0), packetID(0), timestamp(0), payload(0.0), authID(0) {
    // Initialize the error array to zeros
    memset(error, '0', sizeof(error));
}
};


char* PacketToCString(const Packet& packet) {
    // Allocate memory for the C-string. Size depends on your specific needs.
    char* cstr = new char[128];

    sprintf(cstr, "nodeID: %u, packetID: %u, timestamp: %u, payload: %f, error: %s, authID: %u",
            packet.nodeID, packet.packetID, packet.timestamp, packet.payload, packet.error, packet.authID);

    return cstr;
}

// returns packet from Cstring
Packet CStringToPacket(const char* cstr) {
    Packet packet;

    // Use sscanf to parse the string and populate the Packet fields
    int ret = sscanf(cstr, "nodeID: %hhu, packetID: %hu, timestamp: %u, payload: %f, error: %15s, authID:
%hhu",
                     &packet.nodeID, &packet.packetID, &packet.timestamp, &packet.payload, packet.error,
&packet.authID);
    // Optionally, you can check if all fields were successfully parsed
    if (ret != 6) {
        // Handle the error, e.g., by setting default values or logging an error message
    }
    return packet;
}



volatile bool canReadTemp = false;

volatile bool Slave2 = false;
volatile bool Slave3 = false;
volatile bool Slave4 = false;
```

```
volatile bool Master = false;

float node1_temperature = 0;
float node2_temperature = 0;
float node3_temperature = 0;
float node4_temperature = 0;


float tempBuffer[WINDOW_SIZE];  // Circular buffer to hold temperature values
int tempIndex = 0;  // Index to keep track of the oldest temperature value
float tempSum = 0;  // Sum of temperatures in the buffer
int tempCount = 0;  // Count of temperatures added to the buffer
int reportCount = 0;  // Count of readings before reporting
float avgTemperature = 0;  // To hold the average temperature
bool newAvgAvailable = false;  // Flag to indicate a new average is available

String jsonTemperature = "{";

int previouse_packet_id = 0;
int current_packet_id = 0;

int LED = 13; //Status LED is on pin 13
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
uint16_t packetCounter = 0;  // Initialize packet counter




//Define functions used in the program
void startTimer(int frequencyHz);
void configureClockTC4();
void configureTimerTC4(int frequencyHz);                //
void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz);
void TC4_Handler();
float Readtemp();
void write_error(int Node_name, int ecode);
void print_errors(char* err);


#define TX_INTERVAL 60000  //Delay between each message in millidecond.

// Pin mapping for SAMD21
const lmic_pinmap lmic_pins = {
    .nss = 12,//RFM Chip Select
    .rxtx = LMIC_UNUSED_PIN,
```

```
    .rst = 7,//RFM Reset
    .dio = {6, 10, 11}, //RFM Interrupt, RFM LoRa pin, RFM LoRa pin
};


TemperatureZero TempZero = TemperatureZero();

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmoc/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

void onEvent (ev_t ev) {
}

osjob_t txjob;
osjob_t timeoutjob;
static void tx_func (osjob_t* job);

// Transmit the given string and call the given function afterwards
void tx(const char *str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1); // Wait a bit, without this os_radio below asserts, apparently because the state hasn't changed
yet
    LMIC.dataLen = 0;
    while (*str)
      LMIC.frame[LMIC.dataLen++] = *str++;
    LMIC.osjob.func = func;
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
    }

    // Enable rx mode and call func when a packet is received
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // RX _now_
    // Enable "continuous" RX (e.g. without a timeout, still stops after
    // receiving a packet)
    os_radio(RADIO_RXON);
    SerialUSB.println("RX");
    }
```

```
static void rxtimeout_func(osjob_t *job) {
    digitalWrite(LED_BUILTIN, LOW); // off
    }

static void rx_func (osjob_t* job) {
    // Blink once to confirm reception and then keep the led on
    digitalWrite(LED_BUILTIN, LOW); // off
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH); // on

    // Timeout RX (i.e. update led status) after 3 periods without RX
    os_setTimedCallback(&timeoutjob, os_getTime() + ms2osticks(3*TX_INTERVAL), rxtimeout_func);

    // Reschedule TX so that it should not collide with the other side's
    // next TX
    os_setTimedCallback(&txjob, os_getTime() + ms2osticks(TX_INTERVAL/2), tx_func);

    SerialUSB.print("Got ");
    SerialUSB.print(LMIC.dataLen);
    SerialUSB.println(" bytes");
    SerialUSB.write(LMIC.frame, LMIC.dataLen);
    SerialUSB.println();

    // Restart RX
    rx(rx_func);
    }

static void txdone_func (osjob_t* job) {
    // delay(1000);
    // rx(rx_func);
    }

    // log text to USART and toggle LED
static void tx_func (osjob_t* job) {
    // say hello
    // tx("First message going in...!", txdone_func);
    // tx(AvhiTemperature.c_str(), txdone_func);
    // reschedule job every TX_INTERVAL (plus a bit of random to prevent
    // systematic collisions), unless packets are received, then rx_func
    // will reschedule at half this time.
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), tx_func);
    }
```

```
    // application entry point
void setup()
{
    SerialUSB.begin(115200);
    // while(!SerialUSB);
    SerialUSB.println("Starting");
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) |
    GCLK_GENCTRL_GENEN |
    GCLK_GENCTRL_SRC_OSCULP32K |
    GCLK_GENCTRL_DIVSEL;
    while(GCLK->STATUS.bit.SYNCBUSY);                                  // wait while synchronizing the GCLK-
>STATUS.bit
                                                                        // WDT clock = clock gen 2
    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT |
    GCLK_CLKCTRL_CLKEN |
    GCLK_CLKCTRL_GEN_GCLK2;
    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0x8; // Set period for chip reset from the datasheet
    WDT->INTENSET.bit.EW = 1; // Enable early warning interrupt
    WDT->CTRL.bit.WEN = 0;
    SerialUSB.begin(115200);
    TempZero.init();


    pinMode(LED_BUILTIN, OUTPUT);

    // initialize runtime env
    os_init();

    // this is automatically set to the proper bandwidth in kHz,
    // based on the selected channel.
    uint32_t uBandwidth;
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    LMIC.freq = 909900000; //change this for assigned frequencies, match with int freq in loraModem.h
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    uBandwidth = 125;
    LMIC.datarate = US915_DR_SF7;         // DR4
    LMIC.txpow = 21;

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    // This sets CR 4/5, BW125 (except for EU/AS923 DR_SF7B, which uses BW250)
```

```
    LMIC.rps = updr2rps(LMIC.datarate);

    SerialUSB.print("Frequency: "); SerialUSB.print(LMIC.freq / 1000000);
              SerialUSB.print("."); SerialUSB.print((LMIC.freq / 100000) % 10);
              SerialUSB.print("MHz");
    SerialUSB.print("  LMIC.datarate: "); SerialUSB.print(LMIC.datarate);
    SerialUSB.print("  LMIC.txpow: "); SerialUSB.println(LMIC.txpow);

    // This sets CR 4/5, BW125 (except for DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    SerialUSB.println("Started");
    SerialUSB.flush();

    // setup initial job
    os_setCallback(&txjob, tx_func);

    // WDT->CTRL.bit.ENABLE = 1;
    startTimer(1);

}


void loop() {
    // execute scheduled jobs and events
    os_runloop_once();
    avgTemperature = Readtemp();

  // Master is transmitting
  if(reportCount == 4)  //&& Master == false
  {


      SerialUSB.println("Sending Master's Average Temperature. ");

      //SAVE THE TEMPERATURE OF NODE 2
      node1_temperature = avgTemperature;

      // Create the packet
      Packet packet;
      packet.nodeID = 2;  // Node 1
```

```
      packet.packetID = packetCounter++;
      packet.timestamp = millis();  // Current time in milliseconds
      packet.payload = avgTemperature;
      // packet.error = 0;
      packet.authID = 10;  // master is sending it's temperature

      // Print and send the message
      SerialUSB.print("SENDING TEMPERATURE OF NODES 2 : PACKET NUMBER :");
      SerialUSB.println(packet.packetID);
      SerialUSB.println();
      SerialUSB.println();



      // jsonTemperature += "\"Avhi\": " + String(node1_temperature, 2) + ", ";
      // jsonTemperature += "\"Amlan\": " + String(node2_temperature, 2) + ", ";
      // jsonTemperature += "\"Shaswati\": " + String(node3_temperature, 2) + ", ";
      // jsonTemperature += "\"Anu\": " + String(node4_temperature, 2);
      // jsonTemperature += "}";

      tx(PacketToCString(packet), txdone_func);
      delay(5000);
      // tx(jsonTemperature.c_str(), txdone_func);

      // SerialUSB.println(jsonTemperature);
      SerialUSB.println();
      SerialUSB.println();
  }
}

void startTimer(int frequencyHz) {
  configureClockTC4();
  configureTimerTC4(frequencyHz);
}
void configureClockTC4() {
  REG_GCLK_CLKCTRL = (uint16_t)(GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_ID_TC4_TC5);
  while (GCLK->STATUS.bit.SYNCBUSY == 1);
}

void configureTimerTC4(int frequencyHz) {
  TcCount16* TC = (TcCount16*)TC4;
  TC->CTRLA.reg &= ~TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  TC->CTRLA.reg |= TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
```

```
  while (TC->STATUS.bit.SYNCBUSY == 1);

  setTimerFrequencyTC4(TC, frequencyHz);
  TC->INTENSET.reg = 0;
  TC->INTENSET.bit.MC0 = 1;

  NVIC_EnableIRQ(TC4_IRQn);
  TC->CTRLA.reg |= TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz) {
  int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
  TC->CC[0].reg = compareValue;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void TC4_Handler() {
  static bool isLEDOn = false;
  TcCount16* TC = (TcCount16*)TC4;
  if (TC->INTFLAG.bit.MC0 == 1)
  {
    TC->INTFLAG.bit.MC0 = 1;
    WDT->CLEAR.reg = WDT_CLEAR_CLEAR_KEY;
    canReadTemp = true;
   digitalWrite(PIN_LED_13, isLEDOn);
   SerialUSB.println(isLEDOn ? "Blue LED is on" : "Blue LED is off");
   isLEDOn = !isLEDOn;
   Master = false;
  }
}

float Readtemp() {
  if (canReadTemp) {
    float temperature = TempZero.readInternalTemperature();

    // Remove the oldest temperature from the sum
    tempSum -= tempBuffer[tempIndex];

    // Add the new temperature to the buffer and sum
    tempBuffer[tempIndex] = temperature;
    tempSum += temperature;

    // Update the index for the oldest temperature
```

```
    tempIndex = (tempIndex + 1) % WINDOW_SIZE;

    // Update the count of temperatures added to the buffer
    if (tempCount < WINDOW_SIZE) {
      tempCount++;
    }

    // Increment the report counter
    reportCount++;

    // If 5 readings have been taken, calculate the average temperature
    if (reportCount >= WINDOW_SIZE) {
      avgTemperature = tempSum / tempCount;
      newAvgAvailable = true;  // Set the flag
      reportCount = 0;  // Reset the report counter
    }

    canReadTemp = false;

  }
  return avgTemperature;
}
```

Slave 3:

```
#define CPU_HZ 48000000
#define TIMER_PRESCALER_DIV 1024
#define WINDOW_SIZE 5  // Size of the sliding window

// #include <sstream>
// #include <string>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <FlashStorage.h>
#include <TemperatureZero.h>

FlashStorage(error_Amlan, int);                // Reserve a portion of flash memory, remove/comment when
upload to the transmitter
FlashStorage(error_Shaswati, int);
FlashStorage(error_Anuruddha, int);
FlashStorage(error_Avhishek, int);
FlashStorage(error_receive, int);
```

```
struct Packet {
  uint8_t nodeID;
  uint16_t packetID;
  uint32_t timestamp;
  float payload;
  char error[16];
  uint8_t authID;

  // Constructor to initialize the values
  Packet() : nodeID(0), packetID(0), timestamp(0), payload(0.0), authID(0) {
    // Initialize the error array to zeros
    memset(error, '0', sizeof(error));
}
};


char* PacketToCString(const Packet& packet) {
    // Allocate memory for the C-string. Size depends on your specific needs.
    char* cstr = new char[128];

    sprintf(cstr, "nodeID: %u, packetID: %u, timestamp: %u, payload: %f, error: %s, authID: %u",
            packet.nodeID, packet.packetID, packet.timestamp, packet.payload, packet.error, packet.authID);

    return cstr;
}

// returns packet from Cstring
Packet CStringToPacket(const char* cstr) {
    Packet packet;

    // Use sscanf to parse the string and populate the Packet fields
    int ret = sscanf(cstr, "nodeID: %hhu, packetID: %hu, timestamp: %u, payload: %f, error: %15s, authID:
%hhu",
                     &packet.nodeID, &packet.packetID, &packet.timestamp, &packet.payload, packet.error,
&packet.authID);
    // Optionally, you can check if all fields were successfully parsed
    if (ret != 6) {
        // Handle the error, e.g., by setting default values or logging an error message
    }
    return packet;
}
```

```
volatile bool canReadTemp = false;

volatile bool Slave2 = false;
volatile bool Slave3 = false;
volatile bool Slave4 = false;
volatile bool Master = false;

float node1_temperature = 0;
float node2_temperature = 0;
float node3_temperature = 0;
float node4_temperature = 0;


float tempBuffer[WINDOW_SIZE];  // Circular buffer to hold temperature values
int tempIndex = 0;  // Index to keep track of the oldest temperature value
float tempSum = 0;  // Sum of temperatures in the buffer
int tempCount = 0;  // Count of temperatures added to the buffer
int reportCount = 0;  // Count of readings before reporting
float avgTemperature = 0;  // To hold the average temperature
bool newAvgAvailable = false;  // Flag to indicate a new average is available

String jsonTemperature = "{";

int previouse_packet_id = 0;
int current_packet_id = 0;

int LED = 13; //Status LED is on pin 13
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
uint16_t packetCounter = 0;  // Initialize packet counter

//Define functions used in the program
void startTimer(int frequencyHz);
void configureClockTC4();
void configureTimerTC4(int frequencyHz);                //
void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz);
void TC4_Handler();
float Readtemp();
void write_error(int Node_name, int ecode);
void print_errors(char* err);


#define TX_INTERVAL 60000  //Delay between each message in millidecond.

// Pin mapping for SAMD21
```

```
const lmic_pinmap lmic_pins = {
    .nss = 12,//RFM Chip Select
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 7,//RFM Reset
    .dio = {6, 10, 11}, //RFM Interrupt, RFM LoRa pin, RFM LoRa pin
};


TemperatureZero TempZero = TemperatureZero();

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmoc/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

void onEvent (ev_t ev) {
}

osjob_t txjob;
osjob_t timeoutjob;
static void tx_func (osjob_t* job);

// Transmit the given string and call the given function afterwards
void tx(const char *str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1); // Wait a bit, without this os_radio below asserts, apparently because the state hasn't changed
yet
    LMIC.dataLen = 0;
    while (*str)
      LMIC.frame[LMIC.dataLen++] = *str++;
    LMIC.osjob.func = func;
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
    }

    // Enable rx mode and call func when a packet is received
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // RX _now_
    // Enable "continuous" RX (e.g. without a timeout, still stops after
    // receiving a packet)
```

```
    os_radio(RADIO_RXON);
    SerialUSB.println("RX");
    }

static void rxtimeout_func(osjob_t *job) {
    digitalWrite(LED_BUILTIN, LOW); // off
    }

static void rx_func (osjob_t* job) {
    // Blink once to confirm reception and then keep the led on
    digitalWrite(LED_BUILTIN, LOW); // off
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH); // on

    // Timeout RX (i.e. update led status) after 3 periods without RX
    os_setTimedCallback(&timeoutjob, os_getTime() + ms2osticks(3*TX_INTERVAL), rxtimeout_func);

    // Reschedule TX so that it should not collide with the other side's
    // next TX
    os_setTimedCallback(&txjob, os_getTime() + ms2osticks(TX_INTERVAL/2), tx_func);

    SerialUSB.print("Got ");
    SerialUSB.print(LMIC.dataLen);
    SerialUSB.println(" bytes");
    SerialUSB.write(LMIC.frame, LMIC.dataLen);
    SerialUSB.println();

    // Restart RX
    rx(rx_func);
    }

static void txdone_func (osjob_t* job) {
    // delay(1000);
    // rx(rx_func);
    }

    // log text to USART and toggle LED
static void tx_func (osjob_t* job) {
    // say hello
    // tx("First message going in...!", txdone_func);
    // tx(AvhiTemperature.c_str(), txdone_func);
    // reschedule job every TX_INTERVAL (plus a bit of random to prevent
    // systematic collisions), unless packets are received, then rx_func
    // will reschedule at half this time.
```

```
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), tx_func);
    }

    // application entry point
void setup()
{
    SerialUSB.begin(115200);
    // while(!SerialUSB);
    SerialUSB.println("Starting");
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) |
    GCLK_GENCTRL_GENEN |
    GCLK_GENCTRL_SRC_OSCULP32K |
    GCLK_GENCTRL_DIVSEL;
    while(GCLK->STATUS.bit.SYNCBUSY);                                   // wait while synchronizing the GCLK-
>STATUS.bit
                                                                         // WDT clock = clock gen 2
    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT |
    GCLK_CLKCTRL_CLKEN |
    GCLK_CLKCTRL_GEN_GCLK2;
    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0x8; // Set period for chip reset from the datasheet
    WDT->INTENSET.bit.EW = 1; // Enable early warning interrupt
    WDT->CTRL.bit.WEN = 0;
    SerialUSB.begin(115200);
    TempZero.init();


    pinMode(LED_BUILTIN, OUTPUT);

    // initialize runtime env
    os_init();

    // this is automatically set to the proper bandwidth in kHz,
    // based on the selected channel.
    uint32_t uBandwidth;
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    LMIC.freq = 909900000; //change this for assigned frequencies, match with int freq in loraModem.h
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    uBandwidth = 125;
    LMIC.datarate = US915_DR_SF7;         // DR4
    LMIC.txpow = 21;

    // disable RX IQ inversion
```

```
    LMIC.noRXIQinversion = true;

    // This sets CR 4/5, BW125 (except for EU/AS923 DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);

    SerialUSB.print("Frequency: "); SerialUSB.print(LMIC.freq / 1000000);
              SerialUSB.print("."); SerialUSB.print((LMIC.freq / 100000) % 10);
              SerialUSB.print("MHz");
    SerialUSB.print("  LMIC.datarate: "); SerialUSB.print(LMIC.datarate);
    SerialUSB.print("  LMIC.txpow: "); SerialUSB.println(LMIC.txpow);

    // This sets CR 4/5, BW125 (except for DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    SerialUSB.println("Started");
    SerialUSB.flush();

    // setup initial job
    os_setCallback(&txjob, tx_func);

    // WDT->CTRL.bit.ENABLE = 1;
    startTimer(1);

}


void loop() {
    // execute scheduled jobs and events
    os_runloop_once();
    avgTemperature = Readtemp();

  // Master is transmitting
  if(reportCount == 4)  //&& Master == false
  {


      SerialUSB.println("Sending Master's Average Temperature. ");

      //SAVE THE TEMPERATURE OF NODE 2
      node1_temperature = avgTemperature;
```

```
      // Create the packet
      Packet packet;
      packet.nodeID = 3;  // Node 1
      packet.packetID = packetCounter++;
      packet.timestamp = millis();  // Current time in milliseconds
      packet.payload = avgTemperature;
      // packet.error = 0;
      packet.authID = 10;  // master is sending it's temperature

      // Print and send the message
      SerialUSB.print("SENDING TEMPERATURE OFNODE 3 : PACKET NUMBER :");
      SerialUSB.println(packet.packetID);
      SerialUSB.println();
      SerialUSB.println();


      tx(PacketToCString(packet), txdone_func);
      delay(5000);

      SerialUSB.println();
      SerialUSB.println();
  }
}

void startTimer(int frequencyHz) {
  configureClockTC4();
  configureTimerTC4(frequencyHz);
}
void configureClockTC4() {
  REG_GCLK_CLKCTRL = (uint16_t)(GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_ID_TC4_TC5);
  while (GCLK->STATUS.bit.SYNCBUSY == 1);
}

void configureTimerTC4(int frequencyHz) {
  TcCount16* TC = (TcCount16*)TC4;
  TC->CTRLA.reg &= ~TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  TC->CTRLA.reg |= TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  setTimerFrequencyTC4(TC, frequencyHz);
  TC->INTENSET.reg = 0;
  TC->INTENSET.bit.MC0 = 1;
```

```
  NVIC_EnableIRQ(TC4_IRQn);
  TC->CTRLA.reg |= TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz) {
  int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
  TC->CC[0].reg = compareValue;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void TC4_Handler() {
  static bool isLEDOn = false;
  TcCount16* TC = (TcCount16*)TC4;
  if (TC->INTFLAG.bit.MC0 == 1)
  {
    TC->INTFLAG.bit.MC0 = 1;
    WDT->CLEAR.reg = WDT_CLEAR_CLEAR_KEY;
    canReadTemp = true;
   digitalWrite(PIN_LED_13, isLEDOn);
   SerialUSB.println(isLEDOn ? "Blue LED is on" : "Blue LED is off");
   isLEDOn = !isLEDOn;
   Master = false;
  }
}

float Readtemp() {
  if (canReadTemp) {
    float temperature = TempZero.readInternalTemperature();

    // Remove the oldest temperature from the sum
    tempSum -= tempBuffer[tempIndex];

    // Add the new temperature to the buffer and sum
    tempBuffer[tempIndex] = temperature;
    tempSum += temperature;

    // Update the index for the oldest temperature
    tempIndex = (tempIndex + 1) % WINDOW_SIZE;

    // Update the count of temperatures added to the buffer
    if (tempCount < WINDOW_SIZE) {
      tempCount++;
```

```
    }

    // Increment the report counter
    reportCount++;

    // If 5 readings have been taken, calculate the average temperature
    if (reportCount >= WINDOW_SIZE) {
      avgTemperature = tempSum / tempCount;
      newAvgAvailable = true;  // Set the flag
      reportCount = 0;  // Reset the report counter
    }

    canReadTemp = false;

  }
  return avgTemperature;
}
```

Slave 4:

```
#define CPU_HZ 48000000
#define TIMER_PRESCALER_DIV 1024
#define WINDOW_SIZE 5  // Size of the sliding window

// #include <sstream>
// #include <string>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <FlashStorage.h>
#include <TemperatureZero.h>

FlashStorage(error_Amlan, int);                 // Reserve a portion of flash memory, remove/comment when
upload to the transmitter
FlashStorage(error_Shaswati, int);
FlashStorage(error_Anuruddha, int);
FlashStorage(error_Avhishek, int);
FlashStorage(error_receive, int);

struct Packet {
  uint8_t nodeID;
  uint16_t packetID;
  uint32_t timestamp;
```

```
  float payload;
  char error[16];
  uint8_t authID;

  // Constructor to initialize the values
  Packet() : nodeID(0), packetID(0), timestamp(0), payload(0.0), authID(0) {
    // Initialize the error array to zeros
    memset(error, '0', sizeof(error));
}
};


char* PacketToCString(const Packet& packet) {
    // Allocate memory for the C-string. Size depends on your specific needs.
    char* cstr = new char[128];

    sprintf(cstr, "nodeID: %u, packetID: %u, timestamp: %u, payload: %f, error: %s, authID: %u",
            packet.nodeID, packet.packetID, packet.timestamp, packet.payload, packet.error, packet.authID);

    return cstr;
}

// returns packet from Cstring
Packet CStringToPacket(const char* cstr) {
    Packet packet;

    // Use sscanf to parse the string and populate the Packet fields
    int ret = sscanf(cstr, "nodeID: %hhu, packetID: %hu, timestamp: %u, payload: %f, error: %15s, authID:
%hhu",
                     &packet.nodeID, &packet.packetID, &packet.timestamp, &packet.payload, packet.error,
&packet.authID);
    // Optionally, you can check if all fields were successfully parsed
    if (ret != 6) {
        // Handle the error, e.g., by setting default values or logging an error message
    }
    return packet;
}



volatile bool canReadTemp = false;

volatile bool Slave2 = false;
volatile bool Slave3 = false;
volatile bool Slave4 = false;
```

```
volatile bool Master = false;

float node1_temperature = 0;
float node2_temperature = 0;
float node3_temperature = 0;
float node4_temperature = 0;


float tempBuffer[WINDOW_SIZE];  // Circular buffer to hold temperature values
int tempIndex = 0;  // Index to keep track of the oldest temperature value
float tempSum = 0;  // Sum of temperatures in the buffer
int tempCount = 0;  // Count of temperatures added to the buffer
int reportCount = 0;  // Count of readings before reporting
float avgTemperature = 0;  // To hold the average temperature
bool newAvgAvailable = false;  // Flag to indicate a new average is available

String jsonTemperature = "{";

int previouse_packet_id = 0;
int current_packet_id = 0;

int LED = 13; //Status LED is on pin 13
long timeSinceLastPacket = 0; //Tracks the time stamp of last packet received
uint16_t packetCounter = 0;  // Initialize packet counter

//Define functions used in the program
void startTimer(int frequencyHz);
void configureClockTC4();
void configureTimerTC4(int frequencyHz);                //
void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz);
void TC4_Handler();
float Readtemp();
void write_error(int Node_name, int ecode);
void print_errors(char* err);


#define TX_INTERVAL 60000  //Delay between each message in millidecond.

// Pin mapping for SAMD21
const lmic_pinmap lmic_pins = {
    .nss = 12,//RFM Chip Select
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 7,//RFM Reset
    .dio = {6, 10, 11}, //RFM Interrupt, RFM LoRa pin, RFM LoRa pin
```

```
};

TemperatureZero TempZero = TemperatureZero();

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmoc/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

void onEvent (ev_t ev) {
}

osjob_t txjob;
osjob_t timeoutjob;
static void tx_func (osjob_t* job);

// Transmit the given string and call the given function afterwards
void tx(const char *str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1); // Wait a bit, without this os_radio below asserts, apparently because the state hasn't changed
yet
    LMIC.dataLen = 0;
    while (*str)
      LMIC.frame[LMIC.dataLen++] = *str++;
    LMIC.osjob.func = func;
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
    }

    // Enable rx mode and call func when a packet is received
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // RX _now_
    // Enable "continuous" RX (e.g. without a timeout, still stops after
    // receiving a packet)
    os_radio(RADIO_RXON);
    SerialUSB.println("RX");
    }

static void rxtimeout_func(osjob_t *job) {
```

```
    digitalWrite(LED_BUILTIN, LOW); // off
    }

static void rx_func (osjob_t* job) {
    // Blink once to confirm reception and then keep the led on
    digitalWrite(LED_BUILTIN, LOW); // off
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH); // on

    // Timeout RX (i.e. update led status) after 3 periods without RX
    os_setTimedCallback(&timeoutjob, os_getTime() + ms2osticks(3*TX_INTERVAL), rxtimeout_func);

    // Reschedule TX so that it should not collide with the other side's
    // next TX
    os_setTimedCallback(&txjob, os_getTime() + ms2osticks(TX_INTERVAL/2), tx_func);

    SerialUSB.print("Got ");
    SerialUSB.print(LMIC.dataLen);
    SerialUSB.println(" bytes");
    SerialUSB.write(LMIC.frame, LMIC.dataLen);
    SerialUSB.println();

    // Restart RX
    rx(rx_func);
    }

static void txdone_func (osjob_t* job) {
    // delay(1000);
    // rx(rx_func);
    }

    // log text to USART and toggle LED
static void tx_func (osjob_t* job) {
    // say hello
    // tx("First message going in...!", txdone_func);
    // tx(AvhiTemperature.c_str(), txdone_func);
    // reschedule job every TX_INTERVAL (plus a bit of random to prevent
    // systematic collisions), unless packets are received, then rx_func
    // will reschedule at half this time.
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), tx_func);
    }

    // application entry point
void setup()
```

```
{
    SerialUSB.begin(115200);
    // while(!SerialUSB);
    SerialUSB.println("Starting");
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) |
    GCLK_GENCTRL_GENEN |
    GCLK_GENCTRL_SRC_OSCULP32K |
    GCLK_GENCTRL_DIVSEL;
    while(GCLK->STATUS.bit.SYNCBUSY);                                  // wait while synchronizing the GCLK-
>STATUS.bit
                                                                        // WDT clock = clock gen 2
    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT |
    GCLK_CLKCTRL_CLKEN |
    GCLK_CLKCTRL_GEN_GCLK2;
    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0x8; // Set period for chip reset from the datasheet
    WDT->INTENSET.bit.EW = 1; // Enable early warning interrupt
    WDT->CTRL.bit.WEN = 0;
    SerialUSB.begin(115200);
    TempZero.init();


    pinMode(LED_BUILTIN, OUTPUT);

    // initialize runtime env
    os_init();

    // this is automatically set to the proper bandwidth in kHz,
    // based on the selected channel.
    uint32_t uBandwidth;
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    LMIC.freq = 909900000; //change this for assigned frequencies, match with int freq in loraModem.h
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    uBandwidth = 125;
    LMIC.datarate = US915_DR_SF7;          // DR4
    LMIC.txpow = 21;

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    // This sets CR 4/5, BW125 (except for EU/AS923 DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);
```

```
    SerialUSB.print("Frequency: "); SerialUSB.print(LMIC.freq / 1000000);
              SerialUSB.print("."); SerialUSB.print((LMIC.freq / 100000) % 10);
              SerialUSB.print("MHz");
    SerialUSB.print("  LMIC.datarate: "); SerialUSB.print(LMIC.datarate);
    SerialUSB.print("  LMIC.txpow: "); SerialUSB.println(LMIC.txpow);

    // This sets CR 4/5, BW125 (except for DR_SF7B, which uses BW250)
    LMIC.rps = updr2rps(LMIC.datarate);

    // disable RX IQ inversion
    LMIC.noRXIQinversion = true;

    SerialUSB.println("Started");
    SerialUSB.flush();

    // setup initial job
    os_setCallback(&txjob, tx_func);

    // WDT->CTRL.bit.ENABLE = 1;
    startTimer(1);

}


void loop() {
    // execute scheduled jobs and events
    os_runloop_once();
    avgTemperature = Readtemp();

  // Master is transmitting
  if(reportCount == 2)  //&& Master == false
  {
      SerialUSB.println("Sending Node 4's Average Temperature. ");

      //SAVE THE TEMPERATURE OF NODE 2
      node1_temperature = avgTemperature;

      // Create the packet
      Packet packet;
      packet.nodeID = 4;  // Node 1
      packet.packetID = packetCounter++;
      packet.timestamp = millis();  // Current time in milliseconds
      packet.payload = avgTemperature;
      // packet.error = 0;
```

```
      packet.authID = 10;  // master is sending it's temperature

      // Print and send the message
      SerialUSB.print("SENDING TEMPERATURE OFNODE 4 : PACKET NUMBER :");
      SerialUSB.println(packet.packetID);
      SerialUSB.println();
      SerialUSB.println();


      tx(PacketToCString(packet), txdone_func);
      delay(5000);

      SerialUSB.println();
      SerialUSB.println();
  }
}

void startTimer(int frequencyHz) {
  configureClockTC4();
  configureTimerTC4(frequencyHz);
}
void configureClockTC4() {
  REG_GCLK_CLKCTRL = (uint16_t)(GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_ID_TC4_TC5);
  while (GCLK->STATUS.bit.SYNCBUSY == 1);
}

void configureTimerTC4(int frequencyHz) {
  TcCount16* TC = (TcCount16*)TC4;
  TC->CTRLA.reg &= ~TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  TC->CTRLA.reg |= TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
  while (TC->STATUS.bit.SYNCBUSY == 1);

  setTimerFrequencyTC4(TC, frequencyHz);
  TC->INTENSET.reg = 0;
  TC->INTENSET.bit.MC0 = 1;

  NVIC_EnableIRQ(TC4_IRQn);
  TC->CTRLA.reg |= TC_CTRLA_ENABLE;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void setTimerFrequencyTC4(TcCount16* TC, int frequencyHz) {
```

```
  int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
  TC->CC[0].reg = compareValue;
  while (TC->STATUS.bit.SYNCBUSY == 1);
}

void TC4_Handler() {
  static bool isLEDOn = false;
  TcCount16* TC = (TcCount16*)TC4;
  if (TC->INTFLAG.bit.MC0 == 1)
  {
    TC->INTFLAG.bit.MC0 = 1;
    WDT->CLEAR.reg = WDT_CLEAR_CLEAR_KEY;
    canReadTemp = true;
   digitalWrite(PIN_LED_13, isLEDOn);
   SerialUSB.println(isLEDOn ? "Blue LED is on" : "Blue LED is off");
   isLEDOn = !isLEDOn;
   Master = false;
  }
}

float Readtemp() {
  if (canReadTemp) {
    float temperature = TempZero.readInternalTemperature();

    // Remove the oldest temperature from the sum
    tempSum -= tempBuffer[tempIndex];

    // Add the new temperature to the buffer and sum
    tempBuffer[tempIndex] = temperature;
    tempSum += temperature;

    // Update the index for the oldest temperature
    tempIndex = (tempIndex + 1) % WINDOW_SIZE;

    // Update the count of temperatures added to the buffer
    if (tempCount < WINDOW_SIZE) {
      tempCount++;
    }

    // Increment the report counter
    reportCount++;

    // If 5 readings have been taken, calculate the average temperature
    if (reportCount >= WINDOW_SIZE) {
```

```
      avgTemperature = tempSum / tempCount;
      newAvgAvailable = true;  // Set the flag
      reportCount = 0;  // Reset the report counter
    }

    canReadTemp = false;

  }
  return avgTemperature;
}
```