

# CSCE 838 Internet Of Things Lab 4 Report

---

## Group Members:

- Member 1: Anirudh Tangellapalli
- Member 2: Matthew Donsig
- Member 3: Audrey Vazzana
- Member 4: Apala Pramanik

## Table of Contents

---

1. Introduction
2. Procedure
  1. Setting up IoT Hub
  2. Merging LMIC Example with Previous Lab Code
  3. Packet Structure and Gateway Setup
  4. Resources and Acknowledgments
  5. Troubleshooting
3. Results
  1. Code Snippets
  2. Serial Monitor Outputs
  3. Azure Cloud Screenshots and JSON File
4. Testing
5. Appendix

## Introduction

---

In this lab, we aim to set up a LoRa-based communication system, leveraging hardware components like the SparkFun Pro RF and the SparkFun ESP32 1-Channel Gateway. LoRa (Long Range) technology allows low-power, wide-area communication between devices, making it an excellent choice for IoT applications. The SparkFun Pro RF will serve as the end device, sending messages, while the ESP32-based 1-Channel Gateway acts as the bridge between the LoRa network and the cloud via WiFi.

We explore how to install the necessary software libraries, configure the hardware, and establish communication between the gateway and a cloud service, Microsoft Azure IoT Hub. This lab demonstrates real-world IoT development by combining hardware setup, cloud service integration, and wireless data transmission.

## Development Process

---

### Setting up IoT Hub

The lab is divided into three primary stages:

#### 1. Setting up the LoRa Gateway (ESP32 1-Channel Gateway):

- First, we install the necessary software on the SparkFun ESP32 Things board, which includes the ESP32 Arduino Core.
- After installing the core, the board is configured to act as a gateway, capable of connecting to the NU-IoT network through WiFi and communicating via LoRa. We also download the modified ESP 1-ch Gateway code (version 5) and adjust several settings for proper functionality.

#### 2. Configuring Azure IoT Hub for Cloud Communication:

- Once the gateway is configured, the next step involves setting up an Azure IoT Hub, which will serve as the cloud platform for receiving and processing data. We create a student account on Azure, set up the IoT Hub, and link it to our gateway by modifying the code to include the IoT device's connection string.

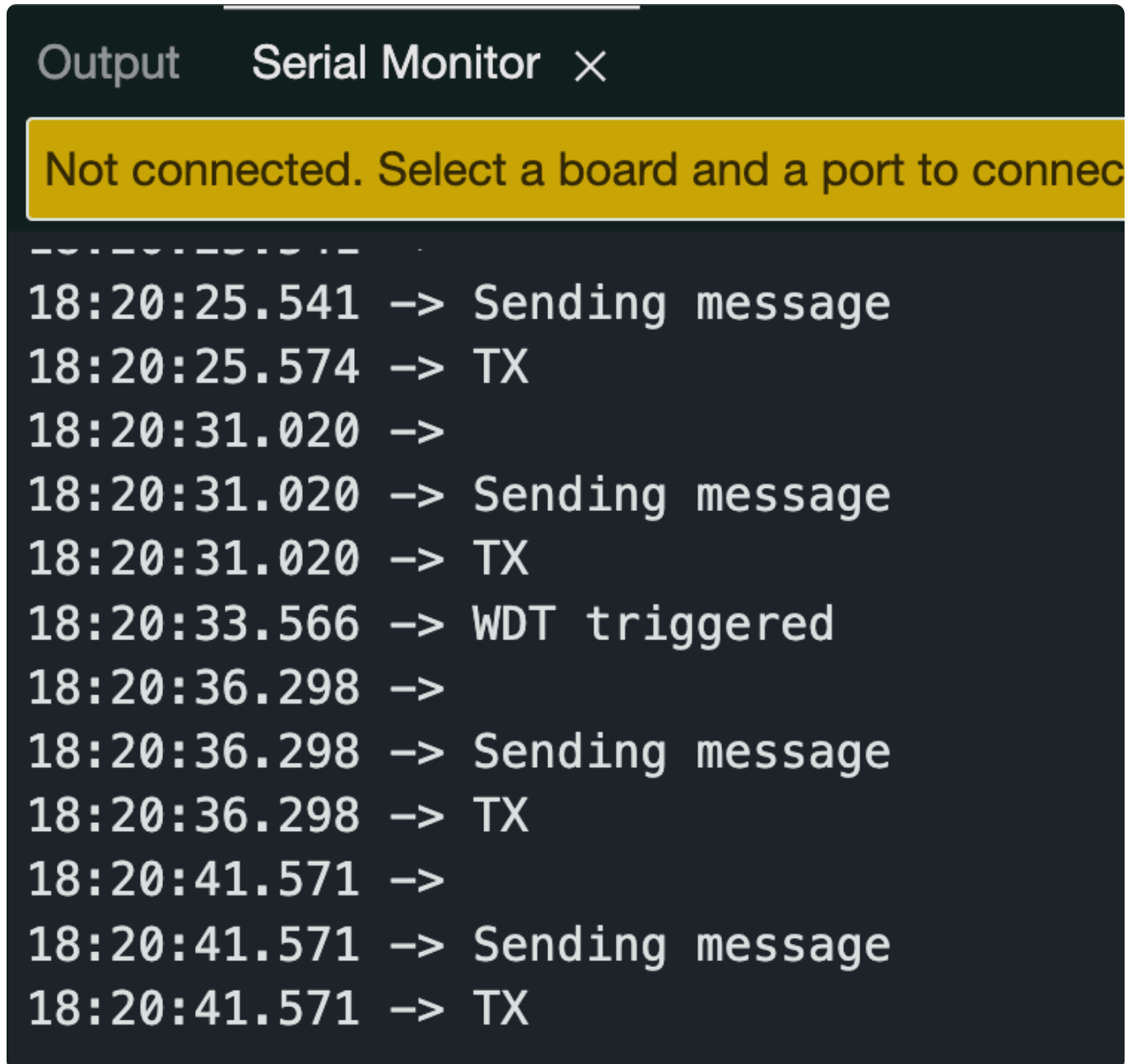
#### 3. Setting Up the End Device (SparkFun Pro RF):

- Finally, we configure the SparkFun Pro RF to act as the end device by installing the LMIC (LoraMAC-in-C) library. The device will transmit data using LoRa, and the messages will be relayed to the Azure cloud through the gateway.

Here are the screenshots showing the process for each team member:

- **Anirudh:**

Serial Monitor Results from "Hello World" code



```
Output  Serial Monitor  X
Not connected. Select a board and a port to connect

-- -- -- -- --
18:20:25.541 -> Sending message
18:20:25.574 -> TX
18:20:31.020 ->
18:20:31.020 -> Sending message
18:20:31.020 -> TX
18:20:33.566 -> WDT triggered
18:20:36.298 ->
18:20:36.298 -> Sending message
18:20:36.298 -> TX
18:20:41.571 ->
18:20:41.571 -> Sending message
18:20:41.571 -> TX
```

## Azure Stream Analytics Query Output

The screenshot shows the Microsoft Azure portal interface for a Stream Analytics job named 'stream\_analytics\_test'. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Job topology, Inputs, Functions, Query (selected), Outputs, and No-code editor (preview). The main area displays the 'Test query' tab with a query language snippet:

```
1 /*
2 Here are links to help you get started with Stream Analytics Query Language:
3 Common query patterns - https://go.microsoft.com/fwlink/?LinkID=619153
4 Query language - https://docs.microsoft.com/stream-analytics-query/query-language-elements-azure-stream-analytics
5 */
```

Below the query, the 'Input preview' tab shows sample events from the 'iotstream' input. The table has the following columns: Message (string), EventProcessedUtcTime (datetime), PartitionId (bigint), EventEnqueuedUtcTime (datetime), and IoT Hub record. The data shows five rows of sample events with timestamps and partition IDs.

Message	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoT Hub record
"Sender = Apala, NodeID = 2, p = 130, Timestamp ..."	"2024-10-09T23:04:08...."	1	"2024-10-09T22:20:55...."	("MessageId":null,"Cor...
"Sender = Apala, NodeID = 2, p = 131, Timestamp ..."	"2024-10-09T23:04:08...."	1	"2024-10-09T22:21:00...."	("MessageId":null,"Cor...
"Sender = Apala, NodeID = 2, p = 132, Timestamp ..."	"2024-10-09T23:04:08...."	1	"2024-10-09T22:21:05...."	("MessageId":null,"Cor...
"Sender = Apala, NodeID = 2, p = 133, Timestamp ..."	"2024-10-09T23:04:08...."	1	"2024-10-09T22:21:10...."	("MessageId":null,"Cor...
"Sender = Apala, NodeID = 2, p = 134, Timestamp ..."	"2024-10-09T23:04:08...."	1	"2024-10-09T22:21:15...."	("MessageId":null,"Cor...

At the bottom, a status message indicates: "While sampling data, no data was received from '1' partitions."

- **Mathew:**  
Serial Monitor Results from "Hello World" code

The screenshot shows a serial monitor window with the following output:

```
18:10:56.150 -> Starting
18:10:56.190 -> Frequency: 902.9MHz IMIC.datarate: 3 IMIC.txpow: 21
18:10:56.191 -> Started
18:10:56.191 -> TX
18:10:56.191 -> test
18:11:06.630 -> TX
18:11:06.630 -> test
18:11:16.860 -> TX
18:11:16.860 -> test
```

## Azure Stream Analytics Query Output

The screenshot shows the Microsoft Azure portal interface for a Stream Analytics job named 'LABFOURStream'. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Job topology, Inputs, Functions, Query (selected), Outputs, and No-code editor (preview). The main area displays the 'Test query' tab with a query language snippet:

```
1 /*
2 Here are links to help you get started with Stream Analytics Query Language:
3 Common query patterns - https://go.microsoft.com/fwlink/?LinkID=619153
4 Query language - https://docs.microsoft.com/stream-analytics-query/query-language-...
5 */
6 SELECT * FROM [MattDon]
7
```

Below the query, the 'Input preview' tab shows sample events from the 'MattDon' input. The table has the following columns: Message (string), EventProcessedUtcTime (datetime), PartitionId (bigint), EventEnqueuedUtcTime (datetime), and IoT Hub record. The data shows two rows of sample events with timestamps and partition IDs.

Message	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoT Hub record
"Apala-Hello there!del..."	"2024-10-09T23:14:59...."	1	"2024-10-09T23:12:17...."	("Me...
"Hello there! -Mathe..."	"2024-10-09T23:14:59...."	1	"2024-10-09T23:12:18...."	("Me...

At the bottom, a status message indicates: "While sampling data, no data was received from '1' partitions."

• **Audrey:**  
Serial Monitor Results from "Hello World" code

```
Starting
Frequency: 902.9MHz  LMIC.datarate: 3  LMIC.txpow: 21
Started
TX Audrey
TX Audrey
TX Audrey
TX Audrey
TX Audrey
---
```

Azure Stream Analytics Query Output

Input previewTest resultsJob simulation (preview)

Showing sample events from 'IoTHubThingGroup1YayAlias'.




TableRaw

RefreshSelect time rangeUpload sample inputDownload sample data






Message	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoTHub
string	datetime	bigint	datetime	record
"You underestimate my pow..."	"2024-10-01T17:17:29.08460..."	1	"2024-10-01T17:16:24.68200..."	{"MessageId":null,"CorrelationId":null,"Connec..."
"You underestimate my pow..."	"2024-10-01T17:17:29.08460..."	1	"2024-10-01T17:17:24.62200..."	{"MessageId":null,"CorrelationId":null,"Connec..."

- **Apala:**

Serial Monitor Results from "Hello World" code



SparkFun SAMD21 Pr... ▾



Lab04\_LMICClient.ino

```
100      os_setTimedCallback(&timeoutjob, os_getT:
101
102      // Reschedule TX so that it should not c
103      // next TX
104      os_setTimedCallback(&txjob, os_getTime()
105
106      SerialUSB.print("Got ");
107      SerialUSB.print(LMIC.dataLen);
108      SerialUSB.println(" bytes");
109      SerialUSB.write(LMIC.frame, LMIC.dataLen
110      SerialUSB.println();
111
112      // Restart RX
```

Output    Serial Monitor ×

Message (Enter to send message to 'SparkFun SAMD21 Pro RF' on 'CC

```
18:17:17.103 -> TX
18:17:17.103 -> TX
18:17:18.431 -> TX
18:17:18.431 -> TX
18:17:19.782 -> TX
18:17:19.782 -> TX
18:17:21.206 -> TX
18:17:21.206 -> TX
18:17:22.527 -> TX
18:17:22.527 -> TX
18:17:23.826 -> TX
18:17:23.826 -> TX
18:17:25.227 -> TX
18:17:25.227 -> TX
18:17:26.554 -> TX
18:17:26.554 -> TX
18:17:27.818 -> TX
18:17:27.818 -> TX
18:17:28.817 -> TX
18:17:28.817 -> TX
18:17:30.193 -> TX
18:17:30.193 -> TX
18:17:31.574 -> TX
18:17:31.574 -> TX
18:17:32.609 -> TX
18:17:32.609 -> TX
18:17:33.697 -> TX
18:17:33.697 -> TX
18:17:35.161 -> TX
18:17:35.161 -> TX
18:17:36.429 -> TX
18:17:36.429 -> TX
18:17:37.793 -> TX
18:17:37.793 -> TX
18:17:39.224 -> TX
18:17:39.224 -> TX
18:17:40.416 -> TX
18:17:40.416 -> TX
```

## Azure Stream Analytics Query Output

Home > ap\_stream

<> ap\_stream | Query ☆ ...

Stream Analytics job

» ▶ Start job 🔗 Open in VS Code 📄 Diagnostic settings 🔄 Refresh 📖 Query language docs ▾ 😊 Share feedback 📖 Tutorial 2 actions required

▼ Inputs (1) +  
    🔗 apala 📄 ...

▼ Outputs (1) +  
    🔗 output 📄

▼ Functions (0) +

▶ Test query 📄 Save query ✕ Discard changes

9 [apala]

Input preview    Test results    Job simulation (preview)

Showing sample events from 'apala'.

📄 Table 📄 Raw 🔄 Refresh ⌚ Select time range ⬆ Upload sample input ⬇ Download sample data

Message <i>string</i>	EventProcessedUtcTime <i>datetime</i>	PartitionId <i>bigint</i>	EventEnqueuedUtcTime <i>datetime</i>	IoTHub <i>record</i>
"Sender = Apala, NodeID = 2, p = 1,..."	"2024-10-09T23:08:30.3072589Z"	0	"2024-10-09T23:00:16.0290000Z"	{"MessageId":null,"CorrelationId":..."
"Hello therella, NodeID = 2, p = 135,..."	"2024-10-09T23:08:30.3072699Z"	0	"2024-10-09T23:00:23.1230000Z"	{"MessageId":null,"CorrelationId":..."
"Sender = Apala, NodeID = 2, p = 1,..."	"2024-10-09T23:08:30.3072811Z"	0	"2024-10-09T23:00:31.6230000Z"	{"MessageId":null,"CorrelationId":..."
"Sender = Apala, NodeID = 2, p = 2,..."	"2024-10-09T23:08:30.3072923Z"	0	"2024-10-09T23:00:37.1070000Z"	{"MessageId":null,"CorrelationId":..."
"Hello therella, NodeID = 2, p = 2, Ti..."	"2024-10-09T23:08:30.3073028Z"	0	"2024-10-09T23:00:55.3260000Z"	{"MessageId":null,"CorrelationId":..."
"Apala-Hello thereIdelD = 2, p = 2, T..."	"2024-10-09T23:08:30.3073138Z"	0	"2024-10-09T23:01:28.1860000Z"	{"MessageId":null,"CorrelationId":..."

## Merging LMIC Example with Previous Lab Code

We merged the LMIC packet transmission example with the code developed in Lab 3:

1. **Remove Radio Operations:** Removed radio transmission code from Lab 3.
2. **Insert LMIC Code:** Integrated the LMIC library and utilized its packet transmission functionality.
3. **Maintain Packet Structure:** Preserved the structure of packets from Lab 3, ensuring compatibility with LMIC.
4. **Average Temperature Reading:** Reused the temperature reading module from Lab 3 to prepare the data for transmission.

## Packet Structure and Gateway Setup

1. The packet structure was adjusted in the gateway to format data into JSON.
2. The packets containing temperature sensor data were sent to the Azure cloud every 60 seconds using LMIC's `TX_INTERVAL`.

## Gateway JSON Packet Example:



```
{  
  "Name",  
  "NodeID",  
  "Pack counter",  
  "Timestamp",  
  "AverageTemperature"  
}
```

## Resources and Acknowledgments

- LMIC library documentation: <https://github.com/mcci-catena/arduino-lmic>  
(<https://github.com/mcci-catena/arduino-lmic>)
- Azure IoT Hub documentation: <https://docs.microsoft.com/en-us/azure/iot-hub/>  
(<https://docs.microsoft.com/en-us/azure/iot-hub/>)
- Stack Overflow for troubleshooting Arduino and LMIC integration issues.

## Troubleshooting

- **Issue:** Encountered errors during packet transmission using LMIC due to mismatched frequencies.
  - **Solution:** Configured the correct frequency band for the region.
- **Issue:** Gateway refused to parse the incoming packet due to incorrect JSON format.
  - **Solution:** Adjusted packet structure to ensure valid JSON before sending data.

## Results

---

### Code Snippets

Each function developed for this lab is explained below.

```
void tx(const char* str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1);           // Wait a bit to ensure state change
    LMIC.dataLen = 0;
    SerialUSB.println("Sending message");

    // Load the string into the LMIC frame
    while (*str)
        LMIC.frame[LMIC.dataLen++] = *str++;

    // Set the function to call after transmission
    LMIC.osjob.func = func;

    // Start the transmission
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
}
```

### Explanation:

This function handles the transmission of data via the LoRa network. It takes a string `str`, loads it into the LoRa module's frame buffer, and sends it. The function `func` is specified as a callback that gets executed after the transmission is complete.

```
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // Set the time for receiving
    os_radio(RADIO_RXON); // Enable continuous receive mode
    SerialUSB.println("RX");
}
```

### Explanation:

This function enables the receiver mode. It listens for incoming packets and calls the specified callback function `func` when a packet is received. It configures the LoRa module to receive without a timeout, making it continuous until a packet is received.

```

static void tx_func(osjob_t* job) {
    float temperature = TempZero.readInternalTemperature(); // Read temperature
    storage[tempCounter++] = temperature;

    if (tempCounter >= 5) {
        tempCounter = 0; // Rolling window for 5 temperature readings
    }

    float sum = 0;
    int count = 0;

    // Sum up the temperatures
    for (int i = 0; i < 5; i++) {
        if (storage[i] != 0) {
            sum += storage[i];
            count++;
        }
    }

    // Calculate average temperature
    float average = count > 0 ? sum / count : 0;

    // Convert the average temperature to string
    char packetToSend[256];
    char avgStr[10];
    floatToStr(avgStr, average);

    // Prepare packet with node ID, counter, and average temperature
    sprintf(packetToSend, "Sender=%s,NodeID=%d,p=%d,Time=%lu,TempAvg=%s", name, node

    // Send the packet
    tx(packetToSend, txdone_func);

    // Reschedule the transmission after a random delay
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), t
}

```

## Explanation:

This function reads the temperature from the sensor, stores it in a circular buffer, calculates the average of the last 5 readings, and transmits the data as a packet. The packet includes the node ID, a packet counter `p`, the time since the device started, and the average temperature. After transmission, it reschedules the job to run again after a random delay to avoid collisions with other transmissions.

```

void startWDT() {
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_GENEN | GCLK_GENCTRL_SRC_C
    while (GCLK->STATUS.bit.SYNCBUSY);

    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT | GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_

    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0xB; // Set WDT period
    WDT->EWCTRL.bit.EWOFFSET = 0xA; // Set early warning offset
    WDT->INTENSET.bit.EW = 1; // Enable WDT interrupt
    WDT->CTRL.bit.WEN = 0; // Watchdog timer enable
    NVIC_EnableIRQ(WDT_IRQn); // Enable WDT interrupt in the NVIC
    WDT->CTRL.bit.ENABLE = 1;
    while (WDT->STATUS.bit.SYNCBUSY);
}

```

### Explanation:

This function configures and starts the Watchdog Timer (WDT) on the SAMD21. The WDT is set to trigger a timeout if the system does not reset it within the configured period. It also configures an early warning interrupt to handle cases where the system may hang.

```

void WDT_Handler() {
    if (WDT->INTFLAG.bit.EW) {
        WDT->INTFLAG.bit.EW = 1; // Clear the early warning flag
        SerialUSB.println("WDT triggered");
    }
}

```

### Explanation:

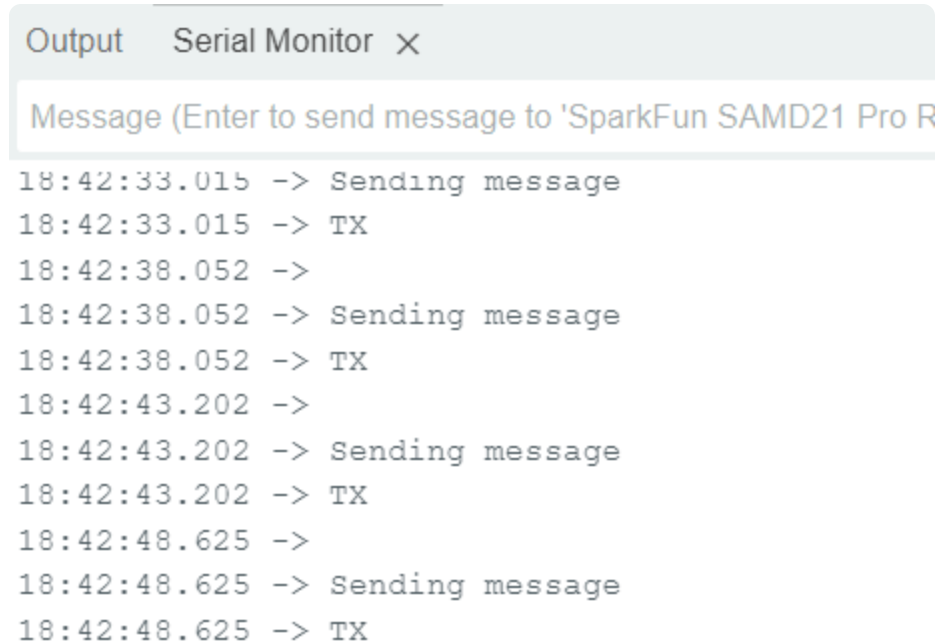
This is the interrupt handler for the Watchdog Timer (WDT). When the WDT triggers its early warning interrupt, this function prints a message to indicate that the WDT was triggered.

These snippets cover the key functionalities of the code, such as LoRa transmission, reception, temperature handling, and the Watchdog Timer.

## Serial Monitor Outputs

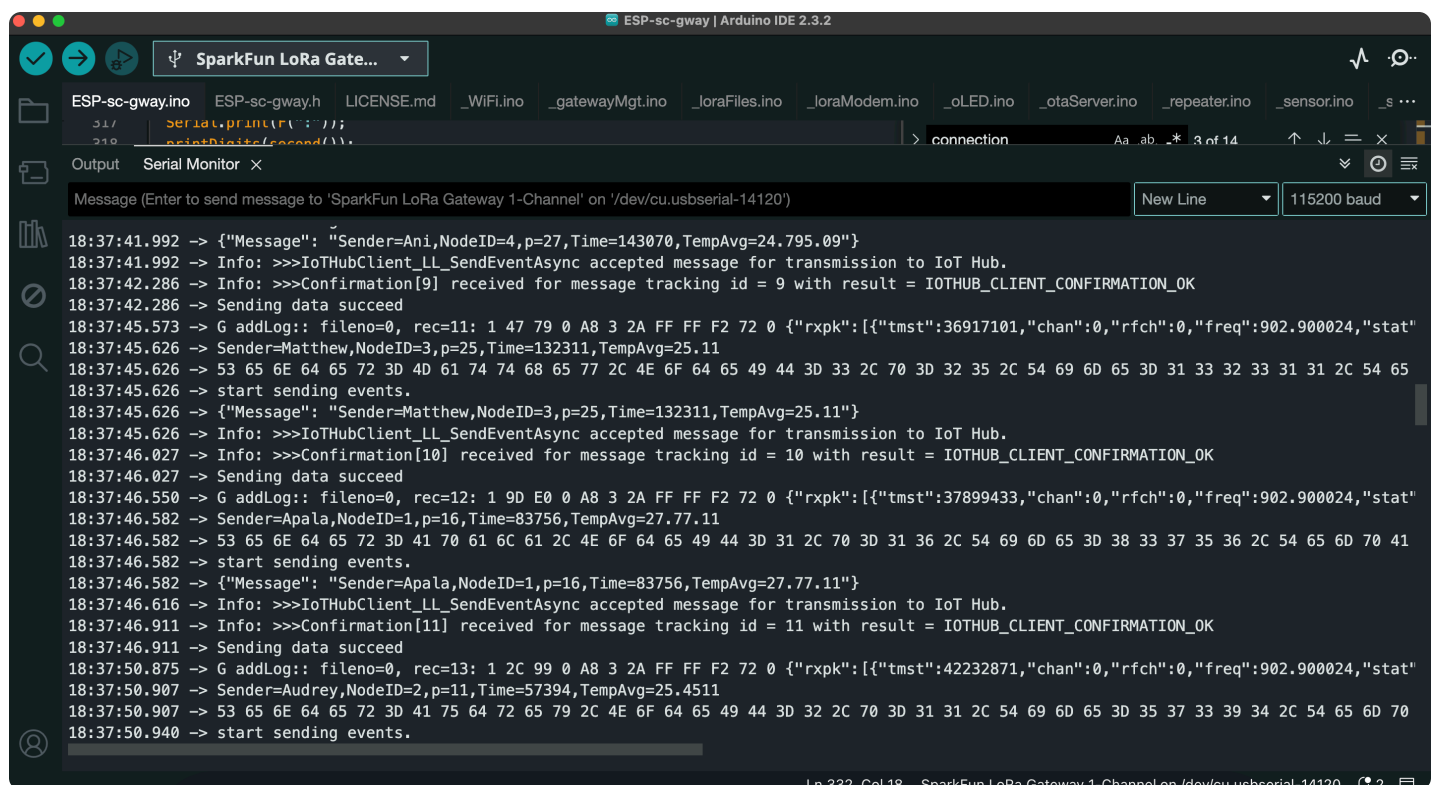
We captured the serial outputs from both the Sparkfun Pro RF device and the LoRa gateway. The outputs confirm successful packet transmission and reception:

## Screenshot of Device Serial Monitor from Leader Node



```
Output  Serial Monitor X
Message (Enter to send message to 'SparkFun SAMD21 Pro R
18:42:33.015 -> Sending message
18:42:33.015 -> TX
18:42:38.052 ->
18:42:38.052 -> Sending message
18:42:38.052 -> TX
18:42:43.202 ->
18:42:43.202 -> Sending message
18:42:43.202 -> TX
18:42:48.625 ->
18:42:48.625 -> Sending message
18:42:48.625 -> TX
```

## Screenshot of GATEWAY Serial Monitor from Leader Node



```
ESP-sc-gway | Arduino IDE 2.3.2
SparkFun LoRa Gate...
ESP-sc-gway.ino  ESP-sc-gway.h  LICENSE.md  _WiFi.ino  _gatewayMgt.ino  _loraFiles.ino  _loraModem.ino  _oLED.ino  _otaServer.ino  _repeater.ino  _sensor.ino  _s...
Serial.print(' ');
Serial.print(recv);
Output  Serial Monitor X
Message (Enter to send message to 'SparkFun LoRa Gateway 1-Channel' on '/dev/cu.usbserial-14120') New Line 115200 baud
18:37:41.992 -> {"Message": "Sender=Ani,NodeID=4,p=27,Time=143070,TempAvg=24.795.09"}
18:37:41.992 -> Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
18:37:42.286 -> Info: >>>Confirmation[9] received for message tracking id = 9 with result = IOTHUB_CLIENT_CONFIRMATION_OK
18:37:42.286 -> Sending data succeed
18:37:45.573 -> G addLog:: fileno=0, rec=11: 1 47 79 0 A8 3 2A FF FF F2 72 0 {"rxpk":{"tmst":36917101,"chan":0,"rfch":0,"freq":902.900024,"stat"
18:37:45.626 -> Sender=Matthew,NodeID=3,p=25,Time=132311,TempAvg=25.11
18:37:45.626 -> 53 65 6E 64 65 72 3D 4D 61 74 74 68 65 77 2C 4E 6F 64 65 49 44 3D 33 2C 70 3D 32 35 2C 54 69 6D 65 3D 31 33 32 33 31 31 2C 54 65
18:37:45.626 -> start sending events.
18:37:45.626 -> {"Message": "Sender=Matthew,NodeID=3,p=25,Time=132311,TempAvg=25.11"}
18:37:45.626 -> Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
18:37:46.027 -> Info: >>>Confirmation[10] received for message tracking id = 10 with result = IOTHUB_CLIENT_CONFIRMATION_OK
18:37:46.027 -> Sending data succeed
18:37:46.550 -> G addLog:: fileno=0, rec=12: 1 9D E0 0 A8 3 2A FF FF F2 72 0 {"rxpk":{"tmst":37899433,"chan":0,"rfch":0,"freq":902.900024,"stat"
18:37:46.582 -> Sender=Apala,NodeID=1,p=16,Time=83756,TempAvg=27.77.11
18:37:46.582 -> 53 65 6E 64 65 72 3D 41 70 61 6C 61 2C 4E 6F 64 65 49 44 3D 31 2C 70 3D 31 36 2C 54 69 6D 65 3D 38 33 37 35 36 2C 54 65 6D 70 41
18:37:46.582 -> start sending events.
18:37:46.582 -> {"Message": "Sender=Apala,NodeID=1,p=16,Time=83756,TempAvg=27.77.11"}
18:37:46.616 -> Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
18:37:46.911 -> Info: >>>Confirmation[11] received for message tracking id = 11 with result = IOTHUB_CLIENT_CONFIRMATION_OK
18:37:46.911 -> Sending data succeed
18:37:50.875 -> G addLog:: fileno=0, rec=13: 1 2C 99 0 A8 3 2A FF FF F2 72 0 {"rxpk":{"tmst":42232871,"chan":0,"rfch":0,"freq":902.900024,"stat"
18:37:50.907 -> Sender=Audrey,NodeID=2,p=11,Time=57394,TempAvg=25.4511
18:37:50.907 -> 53 65 6E 64 65 72 3D 41 75 64 72 65 79 2C 4E 6F 64 65 49 44 3D 32 2C 70 3D 31 31 2C 54 69 6D 65 3D 35 37 33 39 34 2C 54 65 6D 70
18:37:50.940 -> start sending events.
```

# Screenshot of Azure Streaming Analytics from Leader Node

Microsoft Azure portal interface showing the 'stream\_analytics\_test' job. The 'Test results' tab is active, displaying a table of sensor data. The table has columns: Message, EventProcessedUtcTime, PartitionId, EventEnqueuedUtcTime, and IoTHub. The data shows multiple entries from different senders (Matthew, Ani, Apala, Audrey) with various timestamps and partition IDs. A status bar at the bottom indicates 'Showing 30 rows from 'output'.'

Message	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoTHub
"Sender=Matthew,NodeID=3,p=32,..."	"2024-10-09T23:41:22.7292477Z"	1	"2024-10-09T23:38:22.1330000Z"	("MessageId":null,"CorrelationId":null...
"Sender=Ani,NodeID=4,p=35,Time=..."	"2024-10-09T23:41:22.7292572Z"	1	"2024-10-09T23:38:23.5700000Z"	("MessageId":null,"CorrelationId":null...
"Sender=Apala,NodeID=1,p=23,Time=..."	"2024-10-09T23:41:22.7292667Z"	1	"2024-10-09T23:38:24.3520000Z"	("MessageId":null,"CorrelationId":null...
"Sender=Matthew,NodeID=3,p=33,..."	"2024-10-09T23:41:22.7292778Z"	1	"2024-10-09T23:38:27.5860000Z"	("MessageId":null,"CorrelationId":null...
"Sender=Audrey,NodeID=2,p=18,Time=..."	"2024-10-09T23:41:22.7292869Z"	1	"2024-10-09T23:38:28.1640000Z"	("MessageId":null,"CorrelationId":null...
"Sender=Ani,NodeID=4,p=36,Time=..."	"2024-10-09T23:41:22.7292965Z"	1	"2024-10-09T23:38:28.7260000Z"	("MessageId":null,"CorrelationId":null...

## Azure Cloud Screenshots and JSON File

The following screenshots show the successful upload of sensor data to the Azure cloud, including the JSON file download:

- Azure Portal - JSON File Download:

output - Notepad

```
File Edit Format View Help
[{"Message": "Sender=Matthew,NodeID=3,p=30,Time=158464,TempAvg=25.17", "EventProcessedUtcTime": "2024-10-09T23:41:22.7278269Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:11.866000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:16.9680000Z"}],
{"Message": "Sender=Ani,NodeID=4,p=34,Time=179415,TempAvg=24.835.19", "EventProcessedUtcTime": "2024-10-09T23:41:22.7291943Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:18.414000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:22.0500000Z"}],
{"Message": "Sender=Ani,NodeID=4,p=35,Time=184568,TempAvg=24.815.21", "EventProcessedUtcTime": "2024-10-09T23:41:22.7292572Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:23.570000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:27.5030000Z"}],
{"Message": "Sender=Audrey,NodeID=2,p=18,Time=94538,TempAvg=25.5725", "EventProcessedUtcTime": "2024-10-09T23:41:22.7292869Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:28.164000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:29.7710000Z"}],
{"Message": "Sender=Matthew,NodeID=3,p=34,Time=179417,TempAvg=25.25", "EventProcessedUtcTime": "2024-10-09T23:41:22.7293147Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:32.805000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:35.1580000Z"}],
{"Message": "Sender=Matthew,NodeID=3,p=35,Time=184570,TempAvg=25.25", "EventProcessedUtcTime": "2024-10-09T23:41:22.7293424Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:37.945000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:39.5580000Z"}],
{"Message": "Sender=Apala,NodeID=1,p=26,Time=137820,TempAvg=28.0375", "EventProcessedUtcTime": "2024-10-09T23:41:22.7293717Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:40.758000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:50.2840000Z"}],
{"Message": "Sender=Apala,NodeID=1,p=27,Time=143072,TempAvg=28.0923", "EventProcessedUtcTime": "2024-10-09T23:41:22.7293995Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:46.008000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:50.2840000Z"}],
{"Message": "Sender=Apala,NodeID=1,p=28,Time=148150,TempAvg=28.1125", "EventProcessedUtcTime": "2024-10-09T23:41:22.7294266Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:51.086000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:38:55.7930000Z"}],
{"Message": "Sender=Matthew,NodeID=3,p=39,Time=205995,TempAvg=25.25", "EventProcessedUtcTime": "2024-10-09T23:41:22.7294726Z", "PartitionId": 1, "EventEnqueuedUtcTime": "2024-10-09T23:38:59.368000", "DeviceGenerationId": "638630453303658046", "EnqueuedTime": "2024-10-09T23:39:04.6690000Z"}]
```

## Testing

Test Step	Expected Results	Achieved Results	Pass/Fail
1. Testing the LoRa Gateway (ESP32)	Gateway powers up and connects to WiFi; logs successful registration with NU-IoT.	Gateway powered on; connected to WiFi; logged registration.	Pass
	LoRa messages from end device are received.	Messages were successfully received and logged.	Pass
	Gateway connects to Azure IoT Hub without issues.	Successfully connected to Azure IoT Hub.	Pass
2. Testing Azure IoT Hub Configuration	Test messages from end device are received at Azure IoT Hub.	Test messages were received and logged correctly.	Pass
	Device registration shows active status in Azure.	Device registered and shows active status.	Pass
	No errors in Azure logs related to device communication.	No errors found in logs during testing.	Pass
3. Testing the End Device (SparkFun Pro RF)	End device powers on and initializes correctly.	Device powered on and initialized without issues.	Pass
	Data transmission is initiated and logged correctly.	Data transmission was initiated and logged successfully.	Pass
	Test data sent is received at the Azure IoT Hub.	Test data received and logged correctly at Azure.	Pass

# Appendix

---

## Entire Arduino Sketch

```

/*****
 * Copyright (c) 2015 Matthijs Kooijman
 * Copyright (c) 2018 Terry Moore, MCCI Corporation
 *
 * Permission is hereby granted, free of charge, to anyone
 * obtaining a copy of this document and accompanying files,
 * to do whatever they want with them without any restriction,
 * including, but not limited to, copying, modification and redistribution.
 * NO WARRANTY OF ANY KIND IS PROVIDED.
 *
 * This example transmits data on hardcoded channel and receives data
 * when not transmitting. Running this sketch on two nodes should allow
 * them to communicate.
 *****/

#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <TemperatureZero.h>
#include <FlashStorage.h>
#include <math.h> // Ensure this is included for pow() function

// we formerly would check this configuration; but now there is a flag,
// in the LMIC, LMIC.noRXIQinversion;
// if we set that during init, we get the same effect.  If
// DISABLE_INVERT_IQ_ON_RX is defined, it means that LMIC.noRXIQinversion is
// treated as always set.
//
// #if !defined(DISABLE_INVERT_IQ_ON_RX)
// #error This example requires DISABLE_INVERT_IQ_ON_RX to be set. Update \
//      lmic_project_config.h in arduino-lmic/project_config to set it.
// #endif

// How often to send a packet. Note that this sketch bypasses the normal
// LMIC duty cycle limiting, so when you change anything in this sketch
// (payload length, frequency, spreading factor), be sure to check if
// this interval should not also be increased.
// See this spreadsheet for an easy airtime and duty cycle calculator:
// https://docs.google.com/spreadsheets/d/1voGAtQAjC1qBmaVuP1ApNKs1ekgUjavHuVQIXyY
#define TX_INTERVAL 5000 //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
};

// 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-lmloc/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;
int p = 0;
static void tx_func(osjob_t* job);

// Temperature related variables
TemperatureZero TempZero = TemperatureZero();
FlashStorage(storedCounter, int);
int tempCounter = 0;
int timeCounter = 0;
int nodeID = 1;        //TODO: your node here
char* name = "Apala";  //TODO: your name here
float storage[5];

void floatToStr(char* buffer, float value, int precision = 2) {
    int integerPart = (int)value; // Ex
    int decimalPart = abs((int)((value - integerPart) * pow(10, precision))); // Ex

    // Format the string as integerPart.decimalPart
    sprintf(buffer, "%d.%02d", integerPart, decimalPart);
}

// 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, appare
    LMIC.dataLen = 0;
    SerialUSB.println("Sending message");
    // SerialUSB.println(str);
}

```

```
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), rxt

    // 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) {
    //kick WDT
```

```
WDT->CLEAR.reg = WDT_CLEAR_CLEAR_KEY;
}

static void tx_func(osjob_t* job) {
    // Read temperature from sensor
    float temperature = TempZero.readInternalTemperature();

    // Debug: Print the current temperature reading
    // SerialUSB.print("Current Temperature: ");
    // SerialUSB.println(temperature);

    // Store temperature in the array
    storage[tempCounter++] = temperature;

    // Reset tempCounter to 0 when it reaches 5, creating a rolling window of 5 values
    if (tempCounter >= 5) {
        tempCounter = 0;
    }

    // Calculate the sum and average of the current temperatures in the storage array
    float sum = 0;
    int count = 0;

    // Debug: Print the stored values in the array
    // SerialUSB.print("Stored Temperatures: ");
    for (int i = 0; i < 5; i++) {
        // SerialUSB.print(storage[i]);
        // SerialUSB.print(" ");
        if (storage[i] != 0) { // Only sum up valid temperatures
            sum += storage[i];
            count++;
        }
    }
    SerialUSB.println(); // Newline for readability

    // Calculate average only if we have valid values
    float average = count > 0 ? sum / count : 0;

    // Debug: Print the calculated average
    // SerialUSB.print("Calculated Average: ");
    // SerialUSB.println(average);

    // Ensure the buffer is large enough
    char packetToSend[256]; // Increased size to prevent overflow

    // Convert the average float value to a string manually
    char avgStr[10]; // Buffer to hold the formatted average as a string
```

```

floatToStr(avgStr, average); // Convert float to string

// Increment the packet counter
p++;

// Concatenate the packet counter value to the message
sprintf(packetToSend, "Sender=%s,NodeID=%d,p=%d,Time=%lu,TempAvg=%s", name, node

// Debug: Print the packet to be sent
// SerialUSB.print("Packet to send: ");
// SerialUSB.println(packetToSend);

// Send the packet
tx(packetToSend, txdone_func);

// Reschedule job every TX_INTERVAL (plus a bit of random to prevent systematic
os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)), t
}

void startWDT() {
    GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(4);
    GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_GENEN | GCLK_GENCTRL_SRC_C
    while (GCLK->STATUS.bit.SYNCBUSY)
        ;
    GCLK->CLKCTRL.reg = GCLK_CLKCTRL_ID_WDT | GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_

    WDT->CTRL.reg = 0;
    WDT->CONFIG.bit.PER = 0xB;
    WDT->EWCTRL.bit.EWOFFSET = 0xA;
    WDT->INTENSET.bit.EW = 1;
    WDT->CTRL.bit.WEN = 0;
    NVIC_EnableIRQ(WDT_IRQn);
    WDT->CTRL.bit.ENABLE = 1;
    while (WDT->STATUS.bit.SYNCBUSY)
        ;
}

void WDT_Handler() {
    if (WDT->INTFLAG.bit.EW) {
        WDT->INTFLAG.bit.EW = 1;
        SerialUSB.println("WDT triggered");
        // logErrorInFlash(ERROR_CODE_WDT);
    }
}

// application entry point

```

```
void setup() {
  SerialUSB.begin(115200);
  //while(!SerialUSB);
  delay(5000);
  SerialUSB.println("Starting");
  // #ifdef VCC_ENABLE
  // // For Pinoccio Scout boards
  // pinMode(VCC_ENABLE, OUTPUT);
  // digitalWrite(VCC_ENABLE, HIGH);
  // delay(1000);
  // #endif

  pinMode(LED_BUILTIN, OUTPUT);

  // Initialize temp sensor
  TempZero.init();
  //setup WDT
  startWDT();
  // initialize runtime env
  os_init();
  // lmic reset values
  LMIC_reset();

  // this is automatically set to the proper bandwidth in kHz,
  // based on the selected channel.
  uint32_t uBandwidth;
  //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  LMIC.freq = 902900000; //change this for assigned frequencies, match with int f
  //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  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);
}

void loop() {
  // execute scheduled jobs and events
  os_runloop_once();
}
```