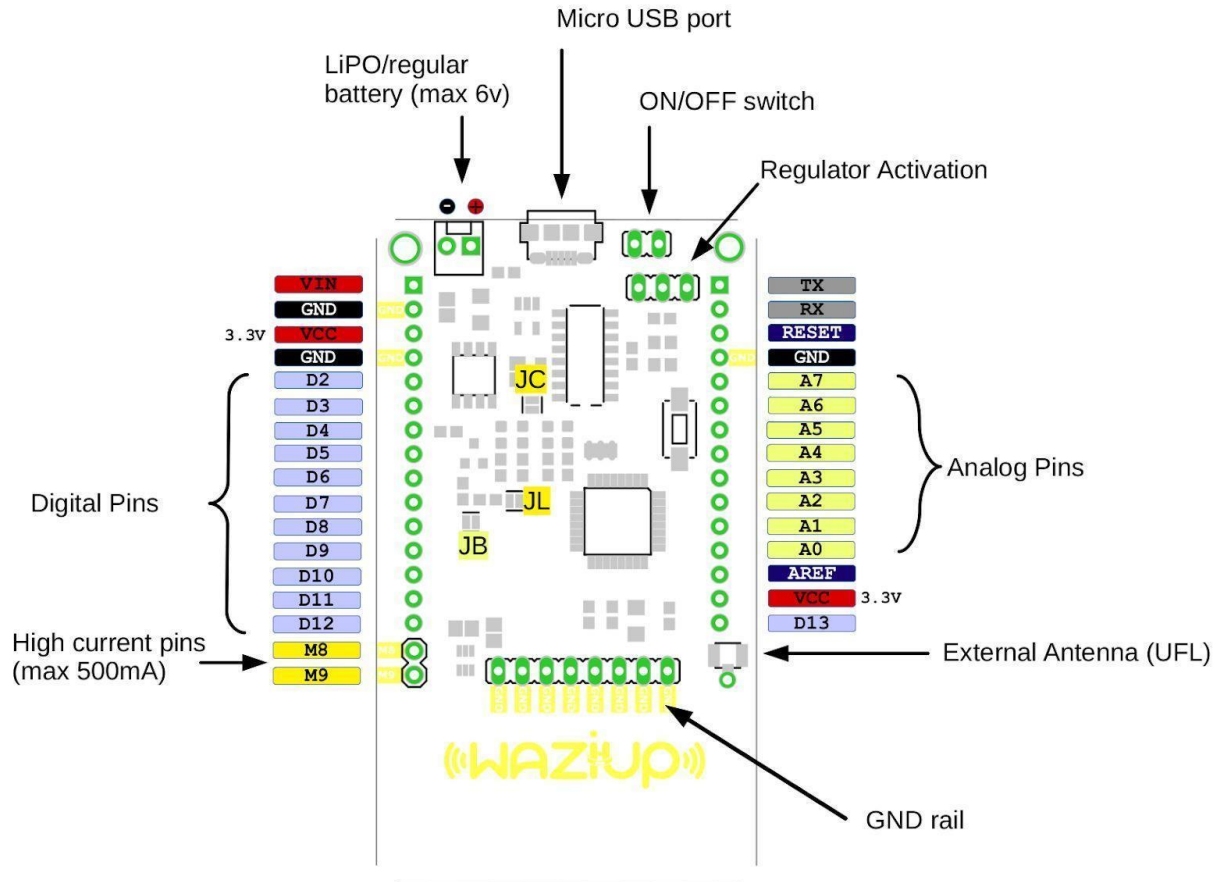


WAZIDEV VERSION 1.3 BOARD

- The WaziDev presents a number of pins where you can connect your sensors and actuators as shown below



PARTS

1. **ON/OFF switch**: This jumper can be used as an on/off switch for the board. It is ON by default.
2. **Regulator Activation**: The jumper indicated by **DIRECT / REG** sets the board to use a regulated voltage or direct. The direct setting is only if you use an input battery of maximum 3.6V any voltage higher than that can damage the LoRa module. **Please note that to always keep this config in REG mode when programming the board.**
3. **Analog Pins**: Arduino standard analog pins A0-A7. Please note that **A7** is connected to the battery voltage level monitoring circuit which can be activated by setting the digital pin **D7** to **LOW**, so **D7** should be set to **HIGH** always.
4. **External Antenna (UFL)**: The board has an embedded PCB antenna which is activated by default and optimized for 868MHz frequency. If you want to use your own antenna instead you need to deactivate the PCB antenna by cutting

its jumper on the back of the board indicated by **JA** then you can connect your antenna to the UFL connector.

5. **High current pins** (*max 500mA*): **M8** and **M9** are high current/voltage programmable output pins. They can be programmed through digital pins **D8** and **D9** respectively. It can be used to activate high current/voltage devices/sensors. The maximum current which can be drained is **500mA** and the maximum voltage is **12v**. The wiring is as follows: The Ground wire of the external high current/voltage source is connected to the same ground of the board (GND), and the positive wire of the power source is connected to the high current device that needs to be controlled by our board. One of the pins of **M8** or **M9** is connected to the Ground of the high current device, then we can turn it on and off by writing **HIGH** and **LOW** to the digital pins **D8** or **D9** respectively.
6. **Digital Pins**: Arduino standard digital pins **D2-D12**. Please note that **D13** is situated on the opposite side of **D12**.
7. **VCC** (*3.3v*): WaziDev board operates with **3.3v**, VCC pins provide 3.3v as output, they can be used as input voltage as well.
8. **Lipo/regular battery**: This port is designed to be used as input for Lithium-Ion rechargeable batteries or just a regular battery. There is an onboard charger which enables the board to use a solar panel. Please note that the solar panel must be connected to either *Micro USB port* or *VIN* pin. **Warning**: your rechargeable battery must have its own protection circuit otherwise it might get overcharged and cause fire. (usually good quality batteries have it)
9. **Micro USB port**: This port is used to power the board on through USB cable and program the board via Arduino IDE.

Jumpers

Below you can find the list of jumpers with their function:

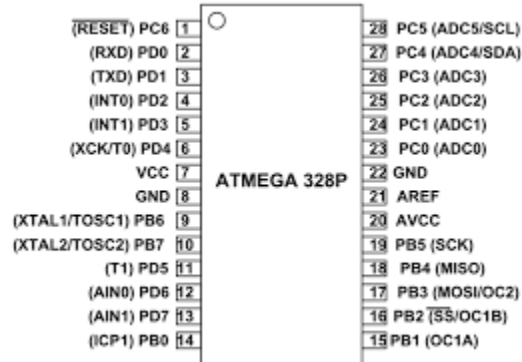
- JL: LED13 and PWR LEDs
- JC: Charger status LEDs (CHG, FULL)
- JB: Battery level read
- JA: Embedded Antenna
- JS: Power Switch
- JR: Radio Interrupt

By default, Jumpers JL, JC, JB, and JA are connected; JS and JR are open. When connected, connects the LoRa interrupt pin to **D2**

Characteristics

Processor System

- MCU: ATmega328p 8Mhz – The WaziDev is powered by an ATmega 328p micro-processor shown below



- RAM: 2 KB
- FLASH: 32 KB

Wireless Network

- Standard: LoRa
- Frequency Band: 863-870MHz for Europe/Africa
- Channels: 1
- Transmit Power: +20dBm -100mW constant RF output
- Receiver Sensitivity: -148dBm
- RF Data Rate: 300kbps
- Modulation: FSK, GFSK, GMSK, MSK, OOK
- Function: Sensor Node
- Antenna connector: Integrated PCB antenna / External UFL

Indicator and Button

- LED: PWR LED, Indicator LED, Charging/Full battery
- Button: 1 reset button
- On/OFF switch: 1, two pins for on/off switch + a jumper to keep the board always on
- Regulator Switch: 1, A jumper that can be used to bypass the regulator for low power applications

I/O

- UART: 1
- ICSP: 1
- I2C: 1
- Analog input: 8 (Arduino standard pins: A0-A7)
- Digital I/O: 9 (Arduino pins, some are used by LoRa)
- Extra GND
- High Current output
- USB

Power

- Supply voltage: 3.3V - 5V 3v (max 3.6v DIRECT and 6v Regulated) Max 1A input current (through Micro USB port)

Environment

- Operational Temperature Operating Humidity: -20 ~ 70 C, 5% ~ 95% Relative Humidity, non-condensing

Mechanical

- Dimensions: 70 x 40 mm

Programming

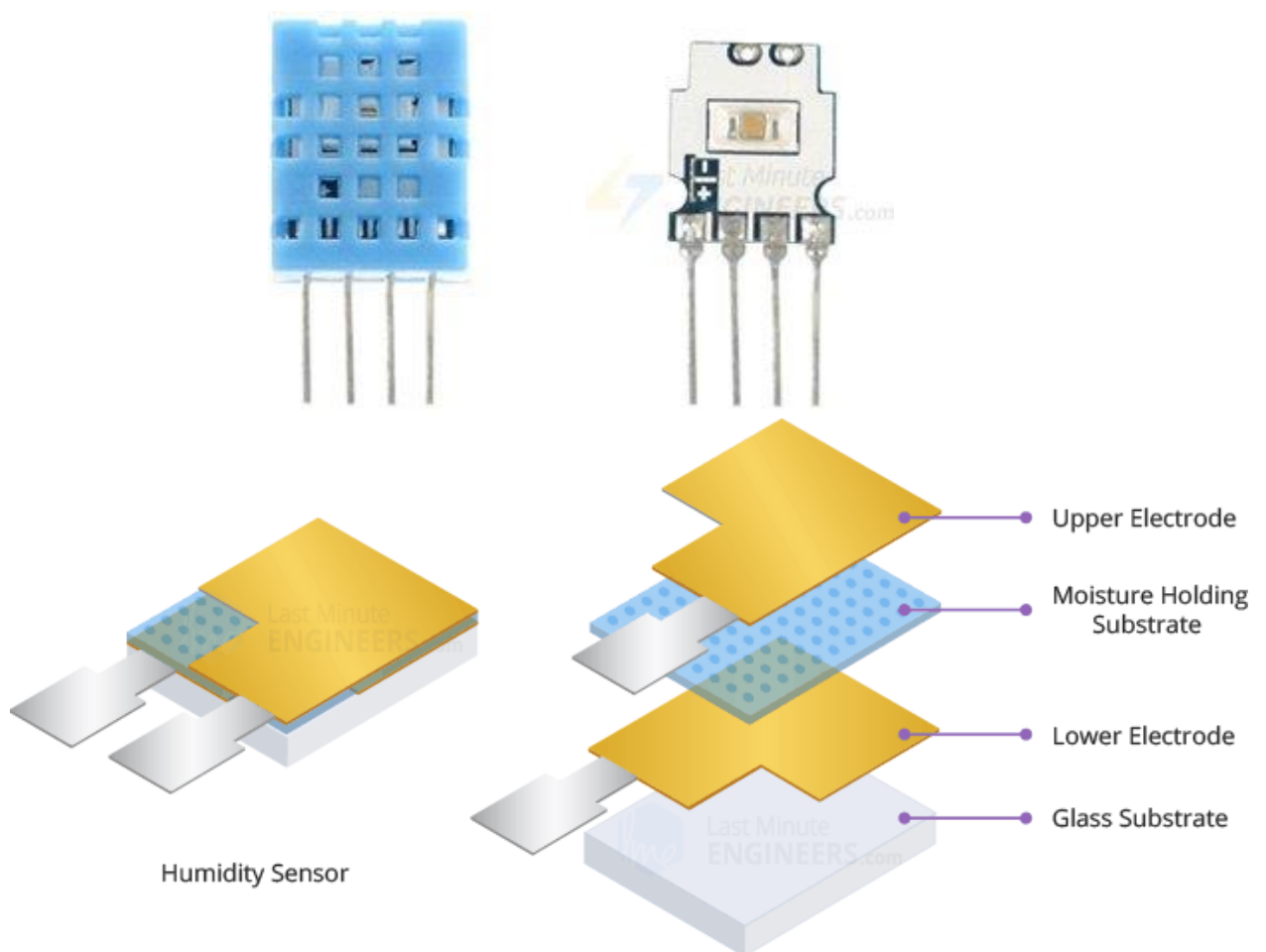
- IDE: Arduino compatible (Select **Pro Mini 3.3V 8Mhz**)

DHT 11

- A DHT 11 Sensor is a basic, ultra-low-cost digital temperature and humidity sensor which uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin.
- It is available as a sensor and a module. The difference between the sensor and module is that the sensor will have a 4-pin package out of which only three pins will be used whereas the module will come with three pins also the module will have a filtering capacitor and pull-up resistor inbuilt and for the sensor you have to use them externally if required.
- To measure the surrounding air this sensor uses a thermistor and a capacitive humidity sensor

Working Principle of a DHT 11 Sensor

- It consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. The ions are released by the substrate as water vapor and absorbed by it, which in turn increases the conductivity between the electrodes

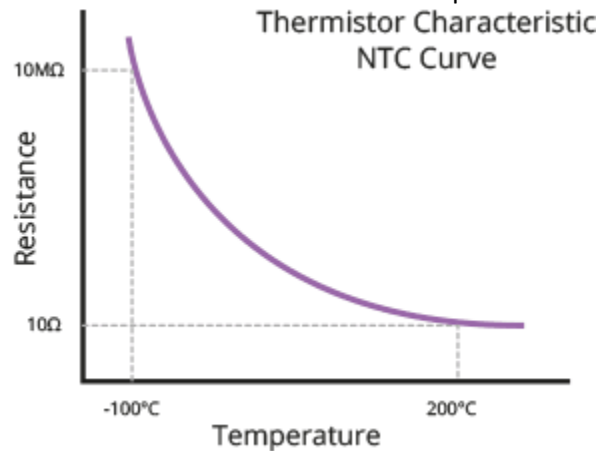


- This change in resistance between the electrodes is proportional to the relative humidity. **Higher relative humidity decreases the resistance between the electrodes, while lower relative humidity increases the resistance between the electrodes.** The IC measures and processes this changed resistance values and changes them to digital form

- For measuring temperature, the sensor uses a Negative Temperature Co-efficient (NTC) thermistor which causes a decrease in its resistance value with increase in temperature.



NTC Thermistor



- To get larger resistance value even for smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

- The temperature range of a DHT 11 is from 0 to 50 degrees Celsius with a 2-degree accuracy.

- The humidity range of this sensor is from 20 to 80% with a 5% accuracy

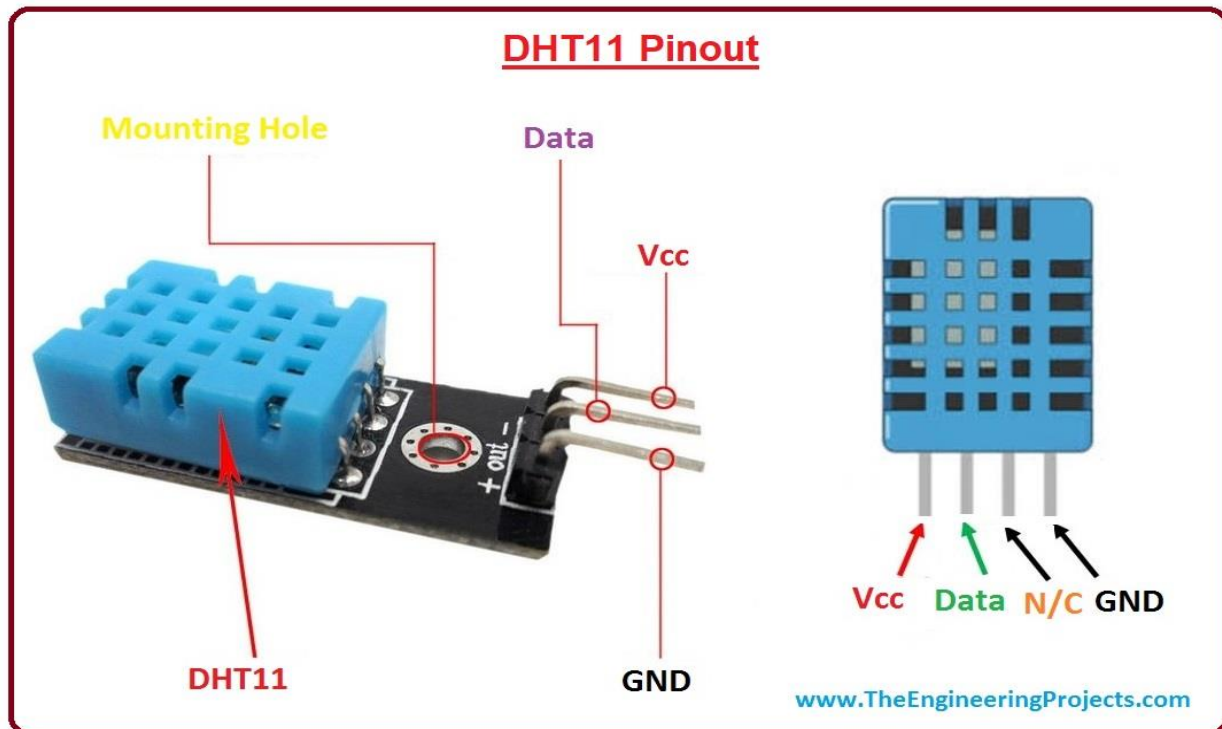
- The sensor has a sampling rate of 1Hz meaning it gives one reading for every second.

- On the other side, there is a small PCB with an 8-bit SOIC -14 packaged IC which measures and processes the analog signal with stored calibration coefficients, does analog to digital conversion and spits out a digital signal with the temperature and humidity.



- The DHT 11 sensor has four pins (sensor module 3 pins) namely VCC, GND, Data Pin and a non-connected pin. The VCC supplies power for the sensor, the out/data pin is used for communication between the sensor and the Arduino

- The GND (Ground Pin) is connected to the ground of the Arduino.



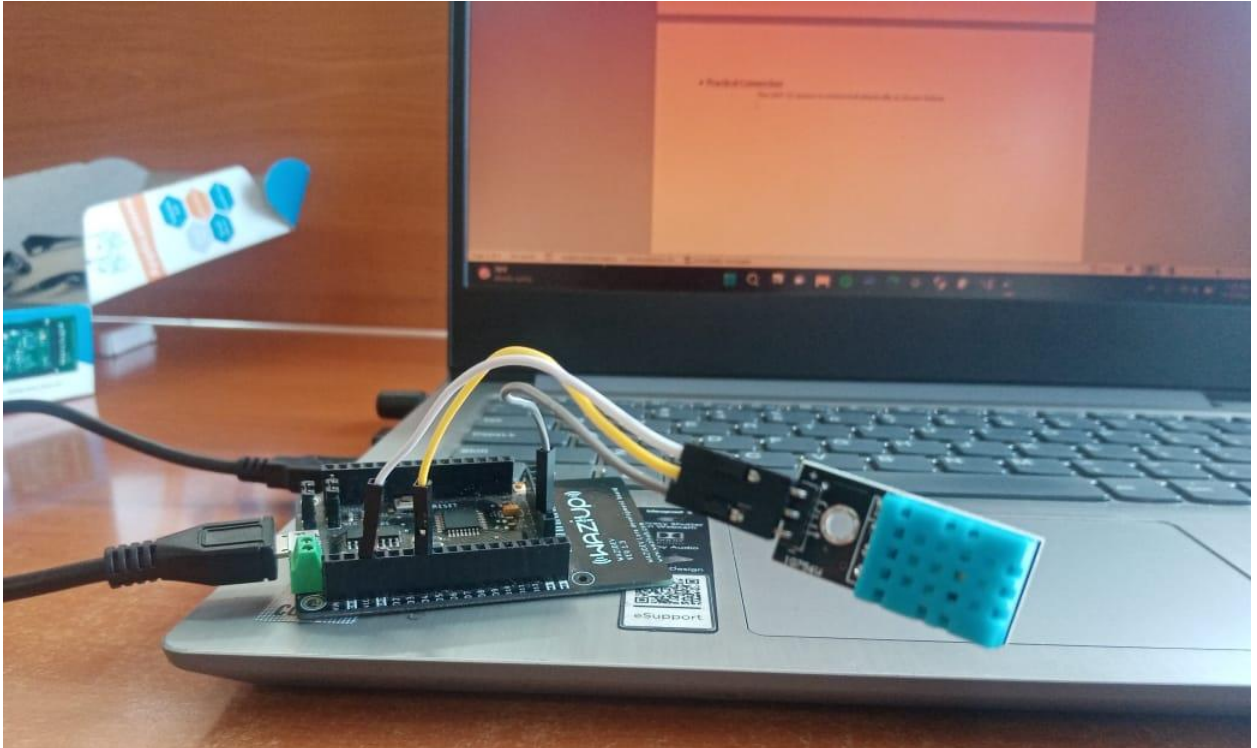
- The DHT 11 Sensor module has essential supporting circuitry hence it should be ready to run without any extra components
- The DHT 11 requires external pull-up resistor of 10 Kilo-ohms between the VCC and the Data pin for proper communication between Sensor and the Arduino. However it has a built-in pull resistor hence no need to use the external pull-up resistor of 10 Kilo-ohms.
- The Module also has a decoupling capacitor for filtering noise on the power supply.

DHT 11 Specifications

- Operating voltage : 3.3V to 5.5V
- Operating Current : 0.3mA(measuring) and 60 micro-amps (Standby)
- Output : Serial Data
- Temperature Range : 0 to 50 degrees Celsius
- Humidity Range : 20% to 80%
- Resolution : Temperature and Humidity are both 16-bit
- Accuracy : $\pm 2^{\circ}\text{C}$ and $\pm 5\%$

Practical Connection

- The DHT 11 sensor is connected physically as shown below where the VCC, Data Pin and the Ground pin have been connected on the WaziDev 1.3 board as shown below



- The code was written on Arduino IDE as shown below then uploaded on to the board to obtain the temperature and Humidity Values.

CODE

//DHT Sensor Code - Temperature and Humidity measurement code

//The first step is to include the library that will be used to carry out the process

//The library used is a DHT Sensor Library by Adafruit

#include "DHT.h"

//The next step is to define the Digital pin where the Data pin of the DHT Sensor will be connected to the board

#define DHTPIN 4

//The next step is to define the type of DHT Sensor we are using

#define DHTTYPE DHT11

//The next step is to define the DHT Parameter so as to initialize the DHT Sensor

DHT dht(DHTPIN, DHTTYPE);

void setup(){

//Inside the void setup, the serial communication and the DHT sensor are initialized

Serial.begin(9600);


```

Serial.println(F("Temperature and Humidity Readings"));

dht.begin();
}

void loop(){

float h = dht.readHumidity(); //Humidity values are read
float t = dht.readTemperature(); //Temperature values are read

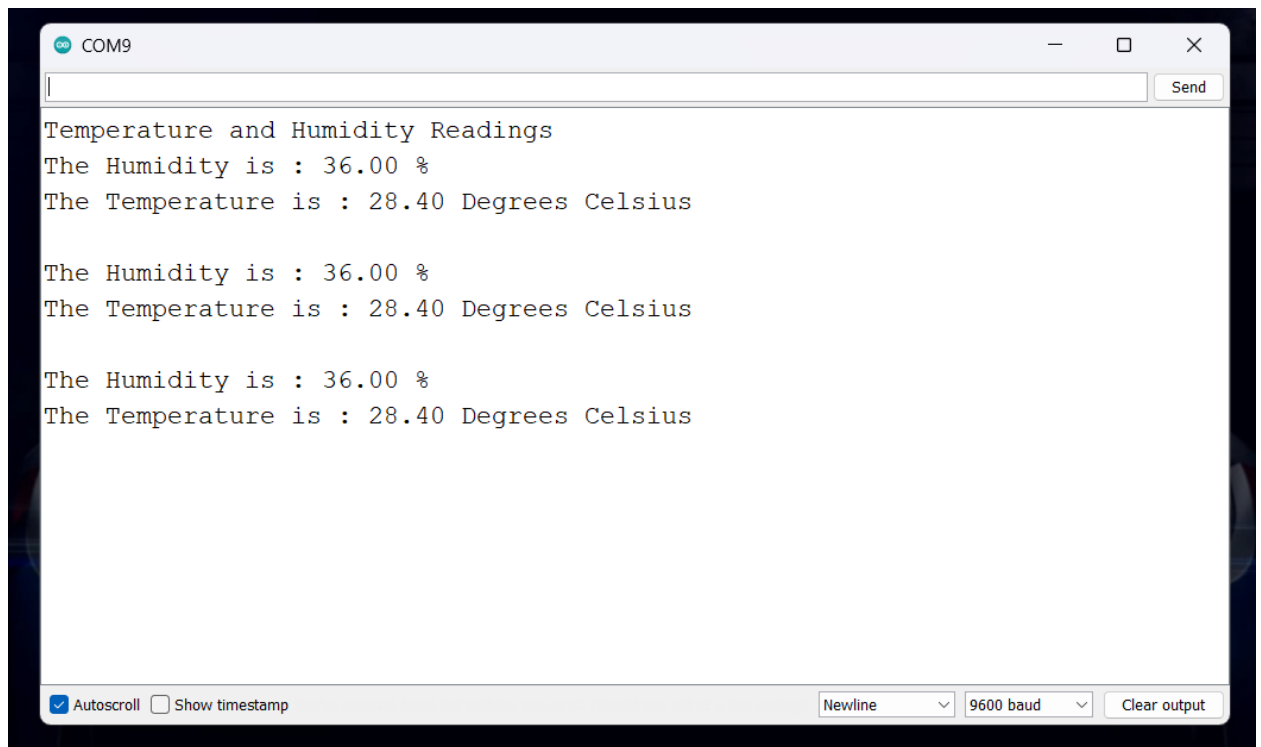
//The Data to be read is checked if any failed and if so the program will exit early and
try to capture the data again
if(isnan(h) || isnan(t)){
Serial.println(F("Failed to capture data from DHT Sensor!"));
return;
}
//Humidity Output Format-The measured humidity value is printed on the serial
monitor
Serial.print(F("The Humidity is : "));
Serial.print(h);
Serial.println(" %");

//Temperature output Format-The measured Temperature value is printed on the
serial monitor
Serial.print(F("The Temperature is : "));
Serial.print(t);
Serial.println(" Degrees Celsius");
Serial.println(" ");

delay(5000); //Wait five seconds before sending another set of data
}

```

- When the code is uploaded onto the Board the following results are obtained on the Serial Monitor



References

- <https://lastminuteengineers.com/dht11-module-arduino-tutorial/#:~:text=The%20DHT11%20sensors%20usually%20require,noise%20on%20the%20power%20supply.>
- <https://www.elprocus.com/a-brief-on-dht11-sensor/>
- <https://components101.com/sensors/dht11-temperature-sensor>

Voltage Level of a battery

- To measure the voltage level of a battery we have to utilize the internal bandgap voltage (Reference voltage).

Measuring the battery voltage by comparing the Vcc voltage with the internal reference voltage using ADC registers

- The first step is to accurately measure the internal reference voltage so as to be able to correctly calibrate the Arduino
- To calibrate the reference you just turn on the reference and measure the voltage at the reference pin

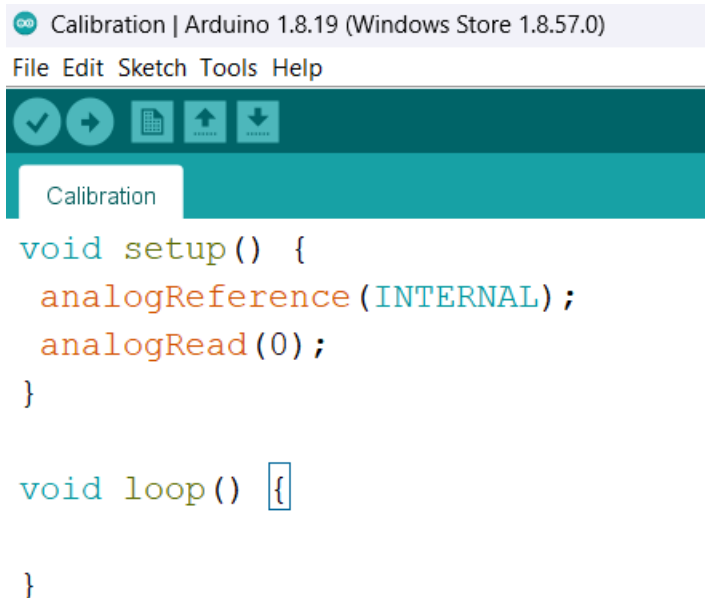
- Turning on the reference is quite simple using the code below

analogReference(INTERNAL)

- - However you will need to activate the ADC to enable the reference so a simple read will do

analogRead(0);

- The above code is placed in the setup() part of the Arduino sketch as shown below

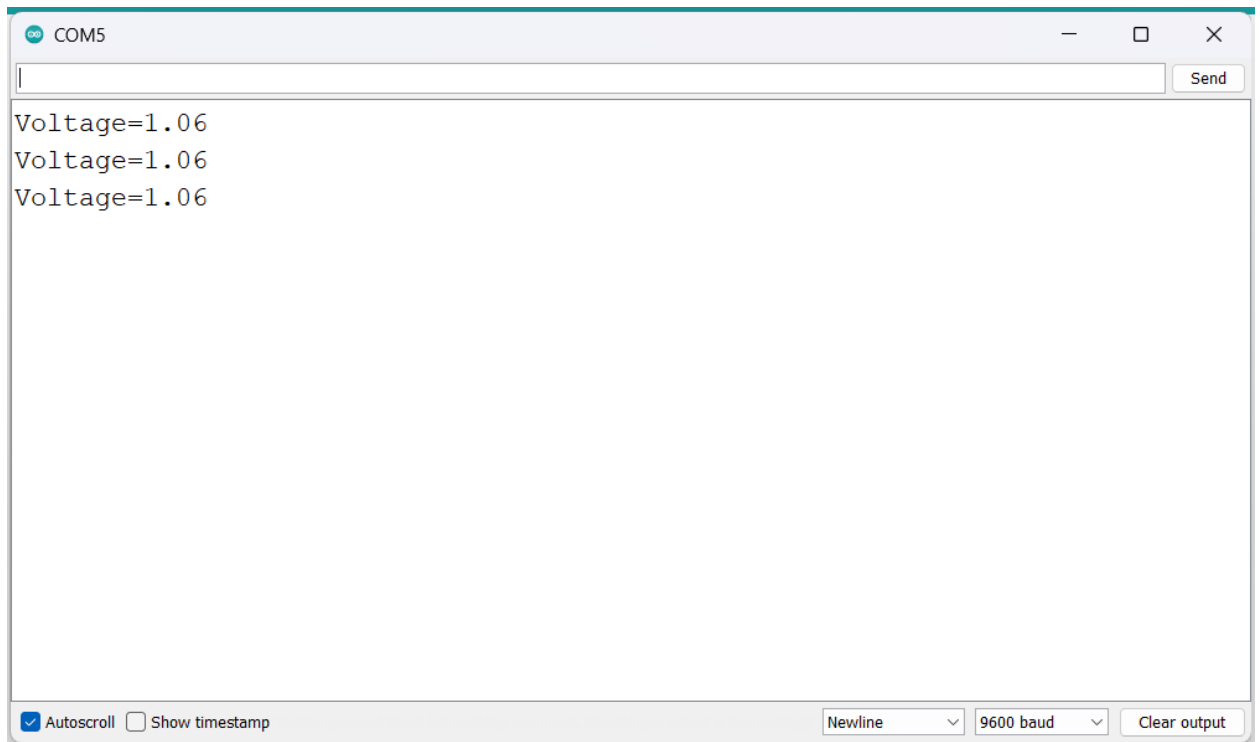


```
Calibration | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help

void setup() {
  analogReference(INTERNAL);
  analogRead(0);
}

void loop() {
}
```

- Now we connect a multimeter to the vref pin and write down the result. In our case we used the Arduino as a multimeter
- The following results were obtained on the Serial monitor of the Arduino



- And thus we can clearly note that our Internal Reference voltage is 1.06Volts
- To set the reference back to supply we use the code below

```
Calibration | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help
[Icons: Checkmark, Run, Serial Monitor, Upload, Download]
Calibration
void setup() {
  analogReference(DEFAULT);
  analogRead(0);
}

void loop() {
  |
}
```

- The code used to carry out the experiment is as shown below

CODE

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    Serial.println(getAccurateVoltage());  
    delay(2000);  
}  
  
int getAccurateVoltage() {  
    readVcc();  
    return readVcc();  
}  
  
int readVcc(void) {  
  
    int result;  
  
    ADCSRA = (1<<ADEN);  
    /*ADEN is the ADC enable Writing this bit to one enables the ADC. By writing it to zero, the  
    ADC is turned off. Turning the ADC off while a conversion is  
    in progress, will terminate this conversion*/  
    ADCSRA |= (1<<ADPS0) | (1<<ADPS1) | (1<<ADPS2);  
    /*These bits determine the division factor between the system clock frequency and input  
    clock to the ADC  
    As per the datasheet, the ADC must be read used with a clock between 50 and 200Khz  
    The system clock frequency for ATmega 328p is 8Megahertz hence when ADPS2:0 are set to  
    111 the division factor is 128 and 8000/128 gives you 62.5Kilo hertz which is within the  
    specified range*/  
  
    // set the reference to Vcc and the measurement to the internal 1.1V reference  
  
    ADMUX = (1<<REFS0) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1);  
    /*REFS1:0 - select the voltage reference of the ADC - when S1 is set to 0  
    and S0 set to 1 the voltage reference used is AVcc with external capacitor at AREF pin  
    MUX3:0 - the value of these bits selects which analog inputs are connected to the ADC  
    when they are set to 1110 respectively the single ended input is 1.1V(Bandgap voltage)  
    Due to the presence of the equal sign the other missing registers in the equation are set to  
    0*/  
  
    delay(1); // Wait for ADC and Vref to settle
```

```

ADCSRA |= (1<<ADSC); // Start conversion
/*ADCSRA=ADCSRA|0100 0000 - Ensure the ADSC is set to 1*/
//ADSC - ADC Start conversion is set to 1 so as to start each conversion
while (bit_is_set(ADCSRA,ADSC)); // wait until done
//Checks whether the ADSC is set to 1 using the while loop if it is 1 conversion occurs
result = ADC;

//The internal voltage reference was measured and obtained to be 1.076V
//The formula is given on the datasheet
//1126400 = 1.076*1024*1000 = 1101824
result = 1101824UL / (unsigned long)result;
return result; // Vcc in millivolts
}

```

- When the code is uploaded on to the board the following results are obtained. An FTDI and Putty were used so as to avoid displaying the charging voltage



```

COM11 - PuTTY
3445
3445
3445
3445
3445
3445
3445
3445
3445
3445
3445

```

- The result displayed is in millivolts hence when converted to volts we obtain

$$3445 \div 1000 = 3.445\text{Volts}$$

REGISTERS AND ANALOG TO DIGITAL CONVERTERS

The ADC uses registers ADMUX, ADCSRA, ADCL, ADCH, ADCSRB, and DIDR0 to configure the hardware and to do analog to digital conversion

The ADC can also be used for measuring battery voltage and current

ADCSRA - ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
(0x7A) (ADCSRA)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPSI	ADPSO
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 - ADEN

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off and turning of the ADC while a conversion is in progress will terminate the conversion

Bit 6 – ADSC

In single conversion mode, write this bit to one to start each conversion.

In free running mode, write this bit to one to start the first conversion

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect

Bit 5 - ADATE - ADC Auto trigger Enable

When this bit is written to one, auto triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC trigger select bits, ADTS in ADCSRB

Bit 4 - ADIF - ADC Interrupt flag

This bit is set when an ADC conversion completes, and the data registers are updated

The ADC conversion complete interrupt is executed if the ADIE bit and the I-bit in SREG are set

ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag

Bit 3 - ADIE - ADC Interrupt Enable

When this bit is written to one and the I-bit in SREG is set, the ADC conversion complete interrupt is activated.

Bit 2:0 - ADPS 2:0 - ADC Prescaler Select Bits

These bits determine the division factor between the system clock frequency and the input clock to the ADC

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADMUX - ADC Multiplexer selection Register

Bit	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7, 6 - REFS1..0: Reference Selection Bits

These bits select the voltage reference for the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversation is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the Aref pin The setting and descriptions are shown in the following table

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF Pin

Bit 5 - ADLAR: ADC Left adjust Result

This bit affects the presentation of the ADC conversion result in the ADC data register.

Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC data register

immediately, regardless of any ongoing conversions

Bit 4 - Res - Reserved Bit

This bit is an unused bit in the Atmel ATmega328P and will always read as zero

Bit 3:0 - MUX3:0 - Analog Channel Selection Bits

The value of this bits select which analog inputs are connected to the ADC

MUX 3:0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8 (1)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (Bandgap voltage VBG)
1111	0V (GND)

References

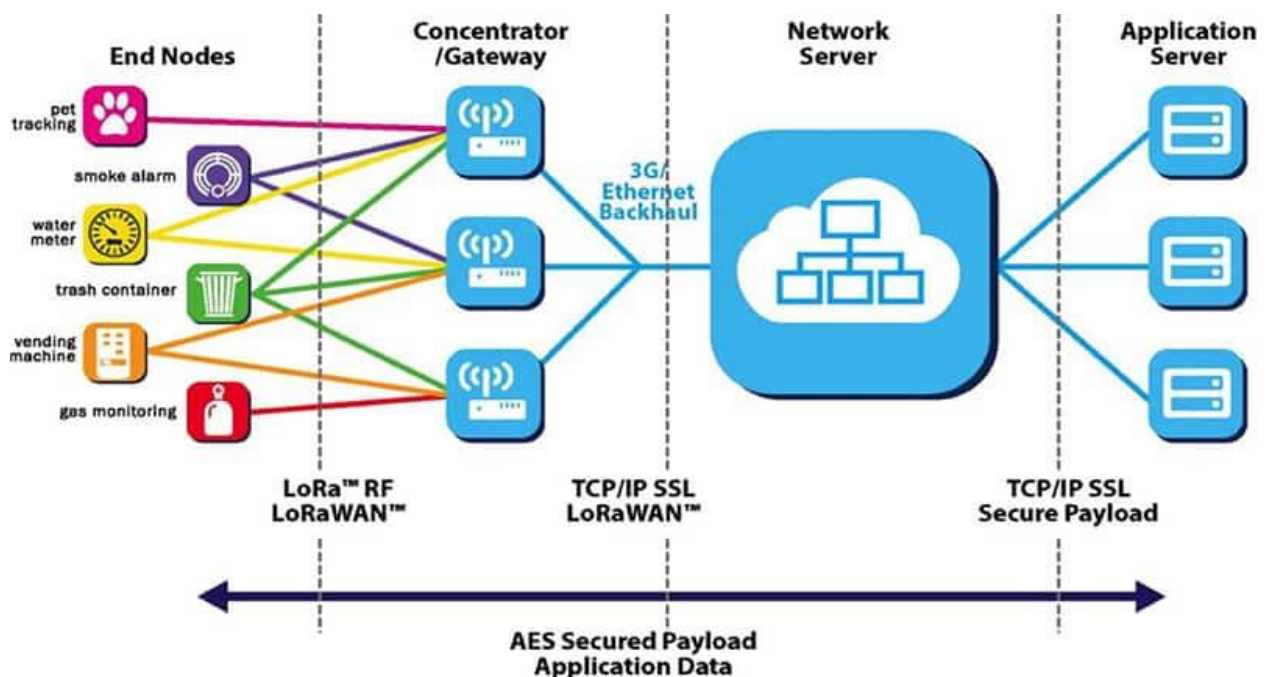
Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet

<https://forum.arduino.cc/t/measure-voltage-of-battery-powering-arduino-pro-mini/880753>

<https://forum.arduino.cc/t/measurement-of-bandgap-voltage/38215>

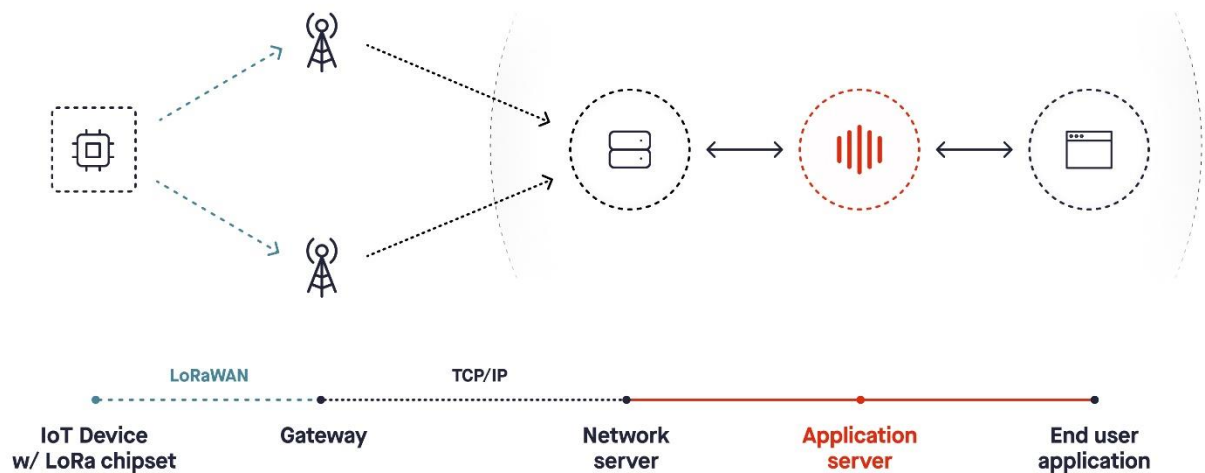
LORA

- LoRa technology was developed by Semtech, and it is a new wireless protocol specifically designed for long-range, low-power communications
- LoRa stands for Long Range Radio and is mainly targeted for M2M and IoT networks
- Each LoRa gateway has the ability to handle up to millions of nodes. The signals can span a significant distance, which means that there is less infrastructure required, making constructing a network much cheaper and faster to implement.
- LoRa also features an adaptive data rate algorithm to help maximize the nodes' battery life and network capacity. The LoRa protocol includes a number of different layers including encryption at the network, application, and device level for secure communications.
- LoRa devices and networks such as LoRaWAN enable smart IOT applications that solve some of the biggest challenges facing our planet such as pollution control etc.



LoRaWAN

- LoRaWAN is described as “a Low Power, Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated things to the internet in regional, national or global networks, and targets key Internet Of Things requirements such as bi-directional communication, end-to-end security, mobility and localization services”
- The LoRaWAN protocol is a Low Power Wide Area Networking (LPWAN) communication protocol that functions on LoRa. The LoRaWAN specification is open so anyone can set up and operate a LoRa network.
- A typical LoRaWAN infrastructure is shown below



Characteristics of LoRaWAN technology

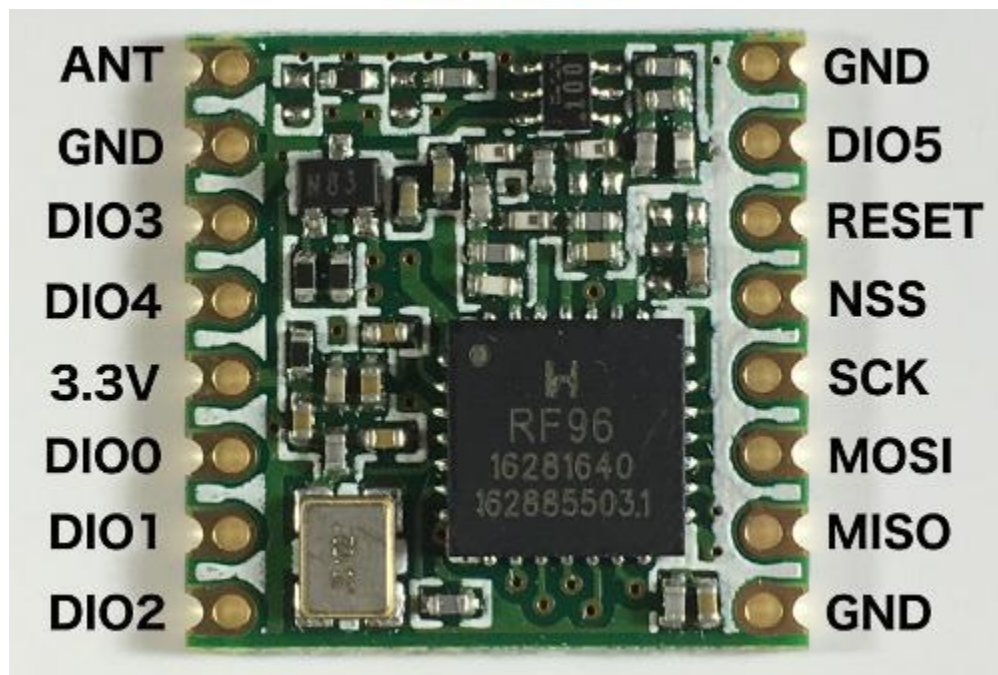
- Long range communication up to 10 miles in line of sight.
- Long battery duration of up to 10 years. For enhanced battery life, you can operate your devices in class A or class B mode, which requires increased downlink latency.
- Low cost for devices and maintenance.
- License-free radio spectrum but region-specific regulations apply.
- Low power but has a limited payload size of 51 bytes to 241 bytes depending on the data rate. The data rate can be 0,3 Kbit/s – 27 Kbit/s data rate with a 222 maximal payload size.

Features of LoRa protocol

Specification	LoRa Feature
Range	2-5Km Urban (1.24-3.1 mi), 15Km suburban (9.3 mi)
Frequency	ISM 868/915 MHz
Standard	IEEE 802.15.4g
Modulation	Spread spectrum modulation type based on FM pulses which vary.
Capacity	One LoRa gateway takes thousands of nodes
Battery	Long battery life
LoRa Physical layer	Frequency, power, modulation and signaling between nodes and gateways

LoRa Modules

- Semtech corporation have introduced a number of LoRa RF modules in the market in particular the SX127x family of RF transceivers for the IoT/M2M markets
- These RF modules operate between 860-1000 MHz and 137-960MHz
- Adafruit has also introduced a number of breakout boards and development boards based around the Semtech RFM69 and RFM95 modules for a range of frequencies such as 433, 868 and 960 Hertz.
- In our case the chip being used is an HopeRFM96W Transceiver module



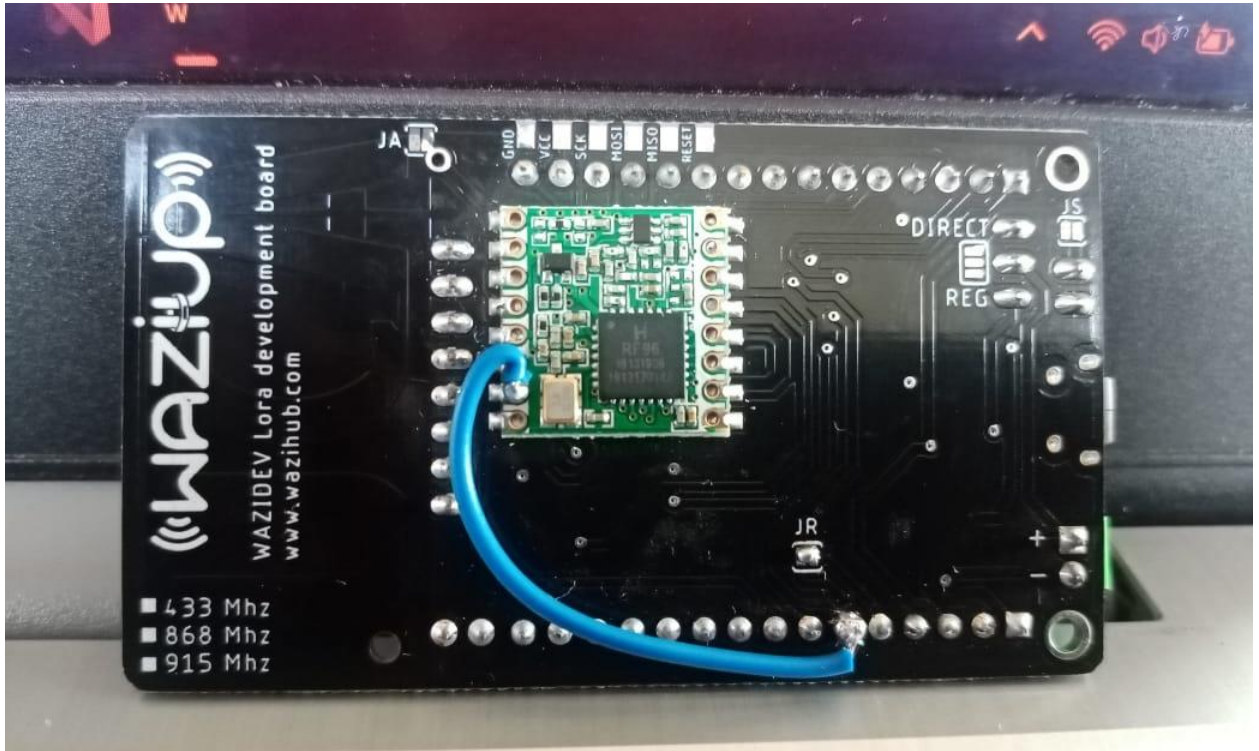
Using Lora Wan with WaziDev V1.3

- The Board used to carry out the illustration is a WaziDev 1.3 LoRa Development board and unlike the WaziDev 1.4 LoRa development board to be able to use LoRaWAN some adjustments are required

Adjustments

- On the WaziDev V 1.3, it is required to solder a small cable between the D3 pin of the WaziDev and the 2nd pin of the LoRa chip, as shown below – soldering the Wire will connect LoRa DIO1 to

the Pro mini pin D3



- It is also required that we close the JR solder pads together with a welding point – Closing the JR will connect LoRa DIO0 pin to the Pro mini pin D2

Requirements

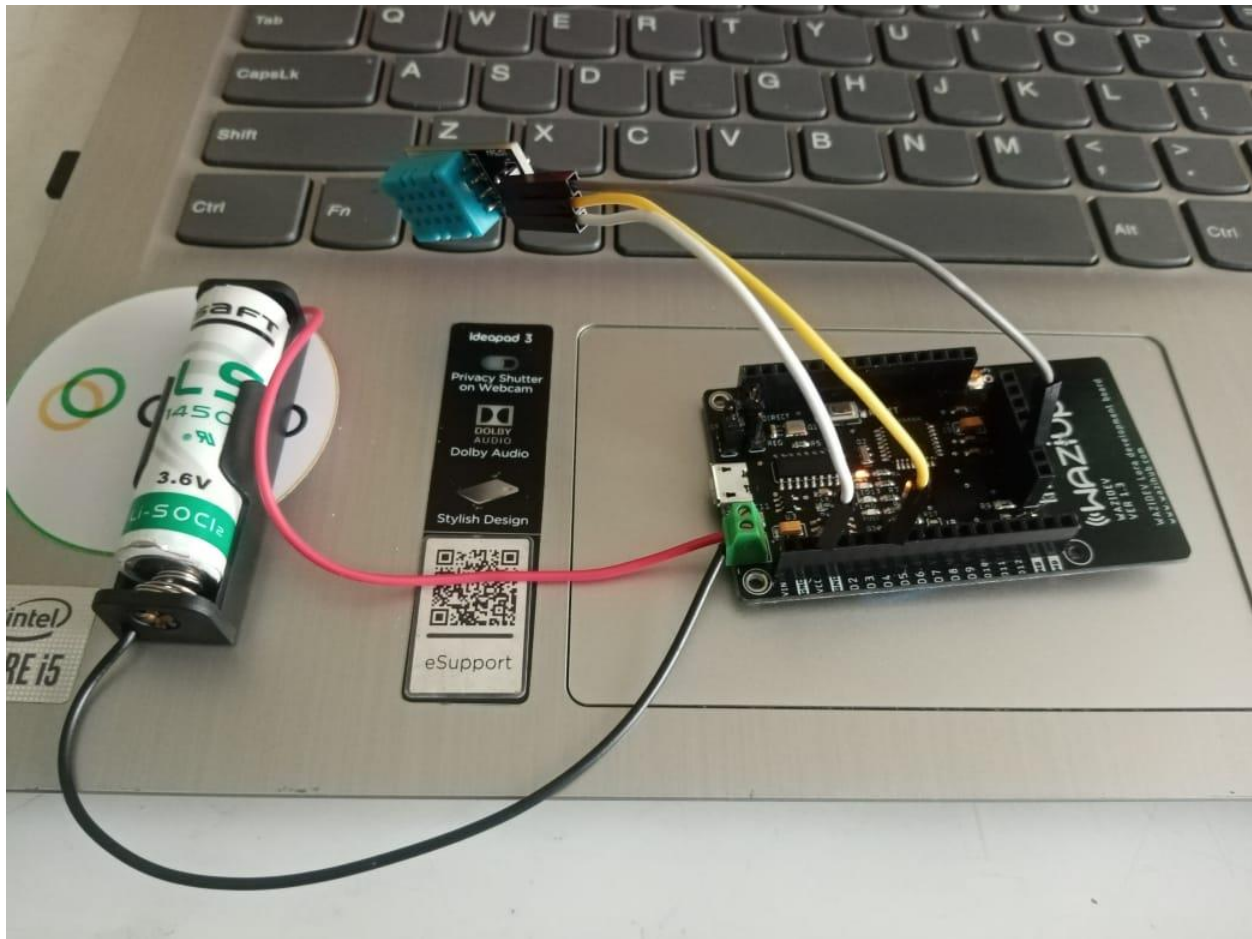
- A welder
- Connecting Cable

Procedure

1. Solder one end of the cable to the D3 pin of the WazirDev
2. Solder the other end of the cable to the 2nd pin of the LoRa Chip
3. Solder the JR pads together

Actual Experiment

- Once the board has been LoRa enabled, the DHT11 sensor module and the battery are connected to the board as shown below



- The Code used to carry out the experiment is shown below where the DHT 11 Temperature and Humidity measurement code and the Battery Voltage measurement code have been added to the IBM (ttn-otaa) code for LoRa Transmission

CODE

//Include the Libraries to be used for the LoRaWAN Transmission

#include<lmic.h>

#include<SPI.h>

#include<hal/hal.h>

//Include the library to be used for the DHT11 Sensor the library is a DHT11 Sensor Library by adafruit

#include "DHT.h"

//Define the Digital pin where the data pin of the DHT sensor will be connected to the board

#define DHTPIN 4

//Define the type of DHT Sensor to be used

#define DHTTYPE DHT11

```

//Define the DHT Parameter so as to initialize the DHT Sensor
DHT dht(DHTPIN, DHTTYPE);

//VOLTAGE LEVEL
int getAccurateVoltage() {
    readVcc();
    return readVcc();
}

int readVcc(void) {

    int result;

    ADCSRA = (1<<ADEN);
    /*ADEN is the ADC enable Writing this bit to one enables the ADC. By writing it to zero, the
    ADC is turned off. Turning the ADC off while a conversion is
    in progress, will terminate this conversion*/
    ADCSRA |= (1<<ADPS0) | (1<<ADPS1) | (1<<ADPS2);
    /*These bits determine the division factor between the system clock frequency and input
    clock to the ADC
    As per the datasheet, the ADC must be read used with a clock between 50 and 200Khz
    The system clock frequency for ATmega 328p is 8Megahertz hence when ADPS2:0 are set to
    111 the division factor is 128 and 8000/128 gives you 62.5Kilo hertz which is within the
    specified range*/

    // set the reference to Vcc and the measurement to the internal 1.1V reference

    ADMUX = (1<<REFS0) | (1<<MUX3) | (1<<MUX2) | (1<<MUX1);
    /*REFS1:0 - select the voltage reference of the ADC - when S1 is set to 0
    and S0 set to 1 the voltage reference used is AVcc with external capacitor at AREF pin
    MUX3:0 - the value of these bits selects which analog inputs are connected to the ADC
    when they are set to 1110 respectively the single ended input is 1.1V(Bandgap voltage)
    Due to the presence of the equal sign the other missing registers in the equation are set to
    0*/

    delay(1); // Wait for ADC and Vref to settle

    ADCSRA |= (1<<ADSC); // Start conversion
    /*ADSCRA=ADSCRA|0100 0000 - Ensure the ADSC is set to 1*/
    //ADSC - ADC Start conversion is set to 1 so as to start each conversion
    while (bit_is_set(ADCSRA,ADSC)); // wait until done
    //Checks whether the ADSC is set to 1 using the while loop if it is 1 conversion occurs
    result = ADC;
    //The internal voltage reference was measured and obtained to be 1.076V

```

```

//The formula is given on the datasheet
//1126400 = 1.076*1024*1000 = 1101824
result = 1101824UL / (unsigned long)result;
return result; // Vcc in millivolts
}

//APPEUI, DEVEUI and APPKEY
//LITTLE ENDIAN FORMAT - The APPEUI should be in little endian format where the LSB comes
first
static const u1_t PROGMEM APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui(u1_t* buf) { memcpy_P(buf, APPEUI, 8); }

//LITTLE ENDIAN FORMAT - The DEVEUI should be in little endian format where the LSB comes
first
static const u1_t PROGMEM DEVEUI[8] = { 0x4F, 0x74, 0x05, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getDevEui(u1_t* buf) { memcpy_P(buf, DEVEUI, 8); }

//BIG ENDIAN FORMAT - The APPKEY should be in big endian format where the MSB comes
first
static const u1_t PROGMEM APPKEY[16] = { 0x92, 0x26, 0x70, 0xD9, 0xBA, 0xAC, 0xB2, 0x09,
0xC9, 0x9F, 0xAF, 0xCD, 0x3D, 0x87, 0x8E, 0x00 };
void os_getDevKey(u1_t* buf) { memcpy_P(buf, APPKEY, 16); }

//Payloads to be sent to the Things Network Gateway
static uint8_t payload[6];
static osjob_t sendjob;

// Schedule TX this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 30;

//Pin Mapping
//The Board used is a WaziDev 1.3 hence the Pin Mapping is as shown below
const lmic_pinmap lmic_pins = {
    .nss = 10, // It is connected to the Micro-controller
    .rxtx = LMIC_UNUSED_PIN, //Unused
    .rst = 4, // It is connected to the Micro-controller reset pin
    .dio = {2,3,LMIC_UNUSED_PIN},
};

void printHex2(unsigned v) {
    v &= 0xff;
    if (v < 16)
        Serial.print('0');

```



```
    Serial.print(v, HEX);  
}
```

```
void onEvent(ev_t ev) {
```

```
    Serial.print(os_getTime());  
    Serial.print(": ");  
    switch (ev) {  
    case EV_SCAN_TIMEOUT:  
        Serial.println(F("EV_SCAN_TIMEOUT"));  
        break;  
    case EV_BEACON_FOUND:  
        Serial.println(F("EV_BEACON_FOUND"));  
        break;  
    case EV_BEACON_MISSED:  
        Serial.println(F("EV_BEACON_MISSED"));  
        break;  
    case EV_BEACON_TRACKED:  
        Serial.println(F("EV_BEACON_TRACKED"));  
        break;  
    case EV_JOINING:  
        Serial.println(F("EV_JOINING"));  
        break;  
    case EV_JOINED:  
        Serial.println(F("EV_JOINED"));  
        {  
            u4_t netid = 0;  
            devaddr_t devaddr = 0;  
            u1_t nwkKey[16];  
            u1_t artKey[16];  
            LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey);  
            Serial.print("netid: ");  
            Serial.println(netid, DEC);  
            Serial.print("devaddr: ");  
            Serial.println(devaddr, HEX);  
            Serial.print("AppSKey: ");  
            for (size_t i = 0; i < sizeof(artKey); ++i) {  
                if (i != 0)  
                    Serial.print("-");  
                printHex2(artKey[i]);  
            }  
            Serial.println("");  
            Serial.print("NwkSKey: ");  
            for (size_t i = 0; i < sizeof(nwkKey); ++i) {
```

```

        if (i != 0)
            Serial.print("-");
        printHex2(nwkKey[i]);
    }
    Serial.println();
}
LMIC_setLinkCheckMode(0);
break;
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));
    break;
case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
    if (LMIC.txrxFlags & TXRX_ACK)
        Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
        Serial.print(F("Received "));
        Serial.print(LMIC.dataLen);
        Serial.print(F(" bytes of payload: "));
    }
    //Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL), do_send);
    break;
case EV_LOST_TSYNC:
    Serial.println(F("EV_LOST_TSYNC"));
    break;
case EV_RESET:
    Serial.println(F("EV_RESET"));
    break;
case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println(F("EV_RXCOMPLETE"));
    break;
case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
    break;
case EV_LINK_ALIVE:
    Serial.println(F("EV_LINK_ALIVE"));
    break;
case EV_TXSTART:
    Serial.println(F("EV_TXSTART"));

```

```

    break;
case EV_TXCANCELED:
    Serial.println(F("EV_TXCANCELED"));
    break;
case EV_RXSTART:
    /* do not print anything -- it wrecks timing */
    break;
case EV_JOIN_TXCOMPLETE:
    Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
    break;

    default:
        Serial.print(F("Unknown event: "));
        Serial.println((unsigned)ev);
        break;
}
}

void do_send(osjob_t* j) {
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    }
    else {
        //Humidity values are read from the DHT11 Sensor and are multiplied by 100 to fit the
        specified range
        uint16_t humidity = dht.readHumidity(false)*100;

        //Temperature values are read from the DHT11 Sensor and are multiplied by 100 to fit the
        specified range
        uint16_t temperature = dht.readTemperature(false)*100;

        //Voltage
        uint16_t voltage = getAccurateVoltage();

        //Bytes are placed into the payload for transmission
        byte payload[6];
        payload[0] = highByte(humidity);
        payload[1] = lowByte(humidity);
        payload[2] = highByte(temperature);
        payload[3] = lowByte(temperature);
        payload[4] = highByte(voltage);
        payload[5] = lowByte(voltage);
    }
}

```

```

    // prepare upstream data transmission at the next possible time.
    // transmit on port 1 (the first parameter); you can use any value from 1 to 223 (others
are reserved).
    // don't request an ack (the last parameter, if not zero, requests an ack from the
network).
    // Remember, acks consume a lot of network resources; don't ask for an ack unless you
really need it.

    LMIC_setTxData2(1, (byte*)payload, sizeof(payload), 0);

    //Measured Temperature, Humidity and Voltage level are displayed on the serial monitor
    Serial.println("The Humidity is : " + String(humidity/100)+" %");
    Serial.println("The Temperature is : " + String(temperature/100)+" degrees celcius");
    Serial.println("The Voltage is : " + String(voltage/1000) + " Volts");
}
// Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
    delay(5000);
    while (! Serial);
    Serial.begin(9600);
    Serial.println(F("Starting"));

    dht.begin();

    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be discarded.
    LMIC_reset();

    //The Arduino Pro mini is running at 8MHz and is relatively slow for LMIC
    //and needs an adjustment by relaxing the timing
    LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);

    // Start job (sending automatically starts OTAA too)
    do_send(&sendjob);
}

void loop() {
    os_runloop_once();
}

```

- Once the code has been uploaded on to the WaziDev 1.3 Board the following Information is displayed on the serial monitor

```

Starting
The Humidity is : 60 %
The Temperature is : 24 degrees celcius
The Voltage is : 3 Volts
318719: EV_JOINING
679655: EV_TXSTART
1001042: EV_JOINED
netid: 19
devaddr: 260BEE37
AppSKey: 11-92-53-92-53-12-70-77-71-A9-A7-6A-7A-A8-E6-36
NwksKey: F4-DB-75-13-E4-78-C7-82-F7-BF-60-2B-CB-82-5E-73
1008367: EV_TXSTART

```

- This shows that the circuit is working and that communication has been established with the Gateway.
- Once transmission has been established, the following kind of information is displayed on the gateway,


dht-sensor-and-voltage-test
ID: dht-sensor-and-voltage-test

↑ 13 ↓ n/a • Last activity 13 seconds ago

Overview Live data Messaging Location Payload formatters Claiming General settings

Time	Type	Data preview
↑ 10:07:14	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 74 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 8.5 RSSI:
↑ 10:07:14	Successfully processed data...	DevAddr: 26 0B 0E 14 <>
↑ 10:06:38	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 74 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 9.8 RSSI:
↑ 10:06:38	Successfully processed data...	DevAddr: 26 0B 0E 14 <>
↑ 10:06:02	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 74 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 9.2 RSSI:
↑ 10:06:02	Successfully processed data...	DevAddr: 26 0B 0E 14 <>
↑ 10:05:26	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 7E 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 8.8 RSSI:
↑ 10:05:26	Successfully processed data...	DevAddr: 26 0B 0E 14 <>
↑ 10:04:49	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 88 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 11 RSSI: -
↑ 10:04:49	Successfully processed data...	DevAddr: 26 0B 0E 14 <>
↑ 10:04:13	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: 17 0C 09 88 0D E2 <> FPort: 1 Data rate: SF7BW125 SNR: 9.2 RSSI:

- The data is in form of payloads hence it has to be formatted so as to make sense
- Once formatted, the data appears as shown below


dht-sensor-and-voltage-test
 ID: dht-sensor-and-voltage-test

↑ 26 ↓ n/a • Last activity just now ⓘ

Overview **Live data** Messaging Location Payload formatters Claiming General settings

Time	Type	Data preview	Verbose stream	Export as JSON	Pause	Clear
↑ 10:14:35	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: { humidity: 59, temperature: 24.2, voltage: 3.554 } 17 0C 09 74 0D E2 <	<input type="checkbox"/>			
↑ 10:14:35	Successfully processed dat...	DevAddr: 26 0B 0E 14 <>				
↑ 10:13:59	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: { humidity: 59, temperature: 24.2, voltage: 3.554 } 17 0C 09 74 0D E2 <	<input type="checkbox"/>			
↑ 10:13:59	Successfully processed dat...	DevAddr: 26 0B 0E 14 <>				
↑ 10:13:23	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: { humidity: 59, temperature: 24.2, voltage: 3.554 } 17 0C 09 74 0D E2 <	<input type="checkbox"/>			
↑ 10:13:23	Successfully processed dat...	DevAddr: 26 0B 0E 14 <>				
↑ 10:12:46	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: { humidity: 60, temperature: 24.2, voltage: 3.554 } 17 70 09 74 0D E2 <	<input type="checkbox"/>			
↑ 10:12:46	Successfully processed dat...	DevAddr: 26 0B 0E 14 <>				
↑ 10:12:10	Forward uplink data message	DevAddr: 26 0B 0E 14 <> Payload: { humidity: 59, temperature: 24.2, voltage: 3.554 } 17 0C 09 74 0D E2 <	<input type="checkbox"/>			
↑ 10:12:10	Successfully processed dat...	DevAddr: 26 0B 0E 14 <>				

- The payload formatter used to carry out the experiment is as shown below

```
function Decoder(bytes, port) {

  var humidity = (bytes[0] <<8) | bytes[1];

  var temperature = (bytes[2] <<8 ) | bytes[3];

  var voltage = (bytes[4] <<8)| bytes[5];

  return{

    humidity: humidity/100,

    temperature: temperature/100,

    voltage: voltage/1000

  }

}
```

```

//Humidity values are read from the DHT11 Sensor and are multiplied by 100 to fit the specified range
uint16_t humidity = dht.readHumidity(false)*100;

//Temperature values are read from the DHT11 Sensor and are multiplied by 100 to fit the specified range
uint16_t temperature = dht.readTemperature(false)*100;

//Voltage
uint16_t voltage = getAccurateVoltage();

//Bytes are placed into the payload for transmission
byte payload[6];
payload[0] = highByte(humidity);
payload[1] = lowByte(humidity);
payload[2] = highByte(temperature);
payload[3] = lowByte(temperature);
payload[4] = highByte(voltage);
payload[5] = lowByte(voltage);

```

- uint16_t is unsigned 16-bit integer hence guarantees 16bits
- Temperature and humidity were multiplied by 100 before packaging hence had to be divided by 100 in the Payload formatter
- The result obtained from the getAccurateVoltage() function is in millivolts hence voltage had to be divided by 1000 in the payload formatter so as to obtain the result in Volts
- Each of the variables to be transmitted was packaged in a high and low byte and since the uint16_t guarantees 16 bits, the high byte took 8bits and the low byte took 8bits

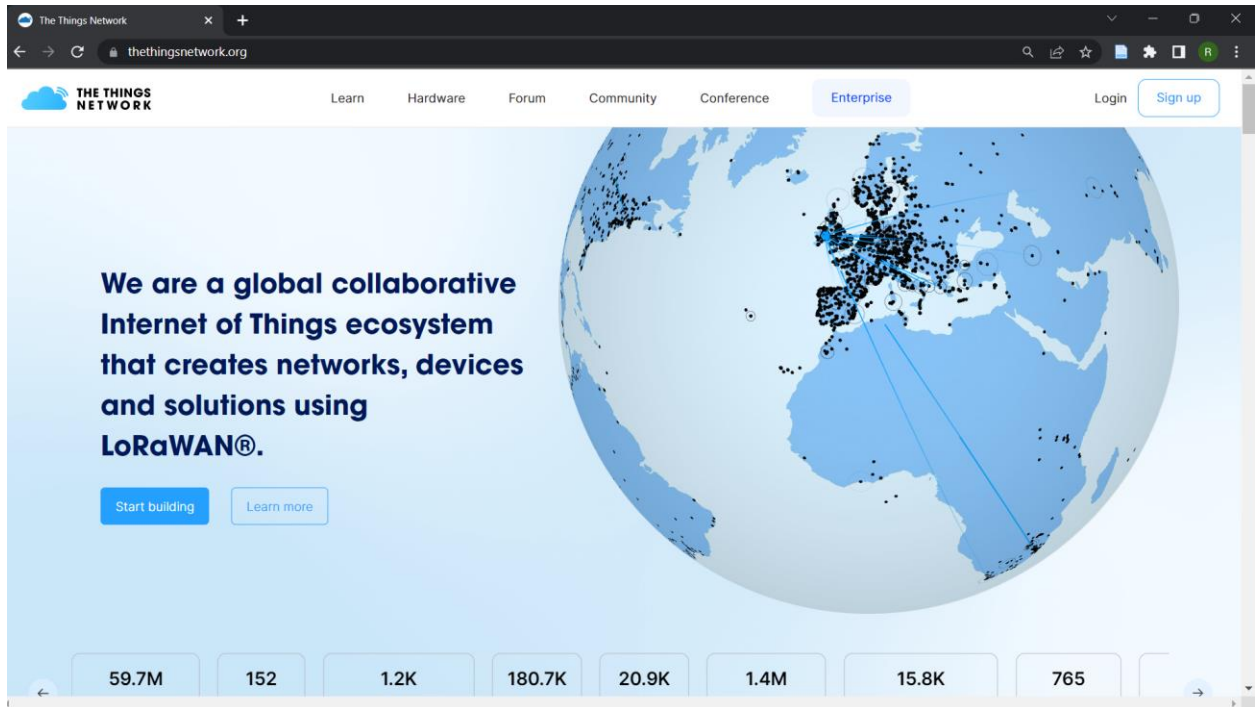
e.g. highByte = 00000000 xxxxxxxx
lowByte = 00000000 yyyyyyyy

- Hence the high byte of each variable had to be shifted to the left eight times in the payload formatter.
- Shifting to the left drops the most significant bit (Left most bit) and adds a zero to the other end
Hence
highByte = xxxxxxxx 00000000
lowByte = 00000000 yyyyyyyy
- Hence when they are added we obtain
xxxxxxx yyyyyyyy
which is the original before packaging.

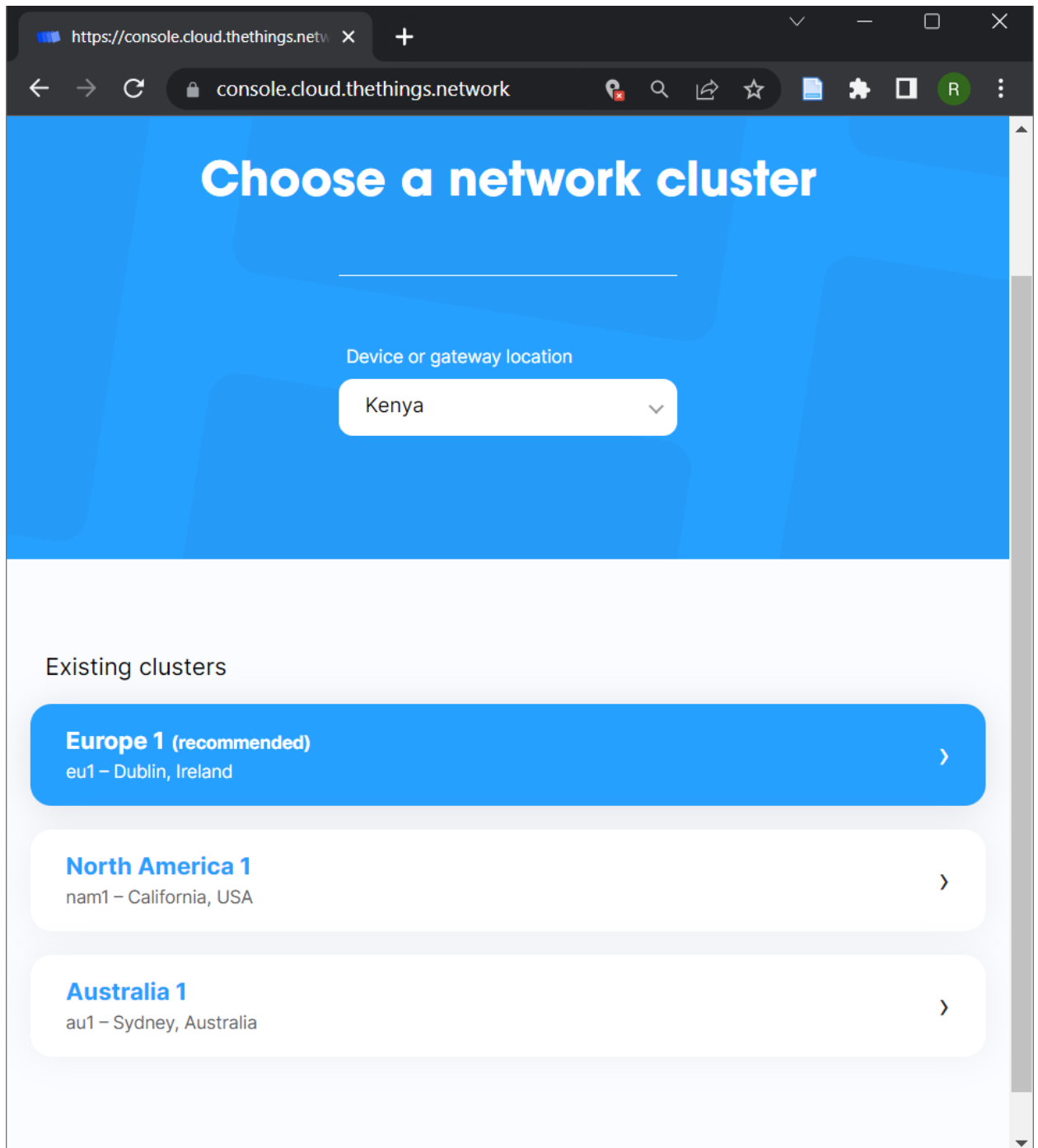
Gateway and Things Network

Creating an account on the things network

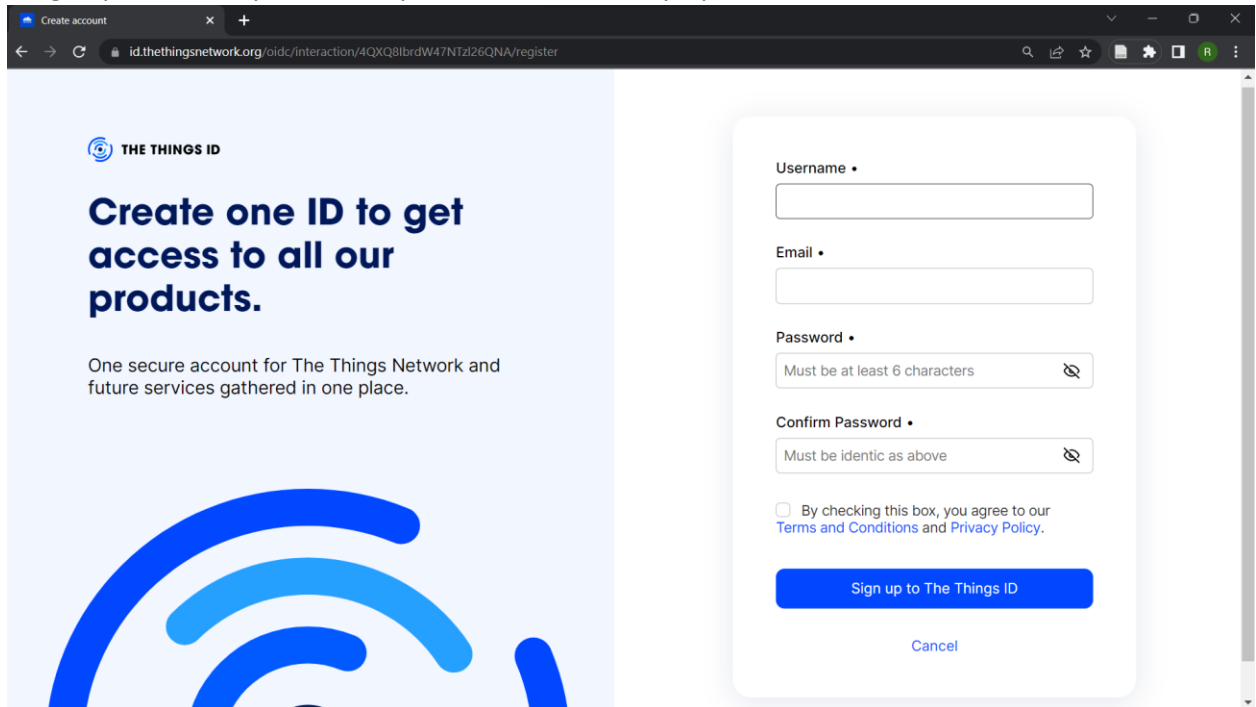
1. Visiting the things network website
<https://www.thethingsnetwork.org/>
2. Select the signup option at the top right part of your screen



3. Select the plan that suits your choice and preference e.g. students etc.
4. Select the location of the device or gateway you are planning on working with after that the recommended network cluster will be recommended to you, select it, and move on



5. A sign-up screen for you to fill in your details will be displayed as shown below



6. Fill in your details and click signup, after which your account will be officially created.

- The MCCI LoRaWAN LMIC library (ttn-otaa example) has a section of including the APPEUI, DEVEUI and the APPKEY as shown below

```
//APPEUI, DEVEUI and APPKEY
//LITTLE ENDIAN FORMAT - The APPEUI should be in little endian format where the LSB comes first
static const u1_t PROGMEM APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui(u1_t* buf) { memcpy_P(buf, APPEUI, 8); }

//LITTLE ENDIAN FORMAT - The DEVEUI should be in little endian format where the LSB comes first
static const u1_t PROGMEM DEVEUI[8] = { 0x4F, 0x74, 0x05, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getDevEui(u1_t* buf) { memcpy_P(buf, DEVEUI, 8); }

//BIG ENDIAN FORMAT - The APPKEY should be in big endian format where the MSB comes first
static const u1_t PROGMEM APPKEY[16] = { 0x92, 0x26, 0x70, 0xD9, 0xBA, 0xAC, 0xB2, 0x09, 0xC9, 0x9F, 0xAF, 0xCD, 0x3D, 0x87, 0x8E, 0x00 };
void os_getDevKey(u1_t* buf) { memcpy_P(buf, APPKEY, 16); }
```

- The APPEUI, DEVEUI and the APPKEY have to be obtained from an end device created in an application inside the things network

Steps of creating an application and an end device in the things network

1. After creating an account, consult your gateway administrator to give you access to the gateway – to add you as a collaborator
2. After which you head to the console section and choose applications as shown below

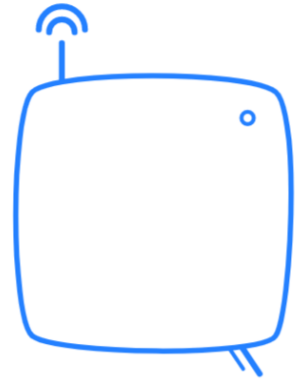
Welcome back, ronny-kimutai! 🙌

Walk right through to your applications and/or gateways.

Need help? Have a look at our [Documentation](#) or [Get support](#).





[Go to applications](#)




[Go to gateways](#)

3. Under applications select the create application button as shown below



[Overview](#) [Applications](#) [Gateways](#) [Organizations](#)

[EU1 Community](#)
Fair use policy applies

 ronny-kimutai

Applications (1)

[+ Create application](#)

ID	Name	End devices	Created at
dht-sensor-and-voltage-test	dht-sensor-and-voltage-test	1	22 days ago

4. In the create application section fill in the application ID, application name and a small description then press next

Create application

Within applications, you can register and manage end devices and their network data. After setting up your device fleet, use one of our many integration options to pass relevant data to your external services.

Learn more in our guide on [Adding Applications](#).

Application ID *

Application name

Description

Optional application description; can also be used to save notes about the application

Create application

5. After creating an application, the following screen appears after which you select the Register end device button

The screenshot displays the 'kimutai-demo' application page in a web interface. The left sidebar contains navigation links: Overview (selected), End devices, Live data, Payload formatters, Integrations, Collaborators, API keys, and General settings. The main content area shows the application details for 'kimutai-demo' (ID: kimutai-demp). It includes a 'General information' section with fields for Application ID, Created at, and Last updated at. A 'Live data' section shows a log entry for '11:01:02 kimutai-de... Create application'. At the bottom, there is a table for 'End devices (0)' with columns for ID, Name, DevEUI, JoinEUI, and Last activity. A search bar and buttons for 'Import end devices' and '+ Register end device' are also present.

Applications > kimutai-demo

kimutai-demo
ID: kimutai-demp

No recent activity

0 End devices 1 Collaborator 0 API keys

General information

Application ID kimutai-demp

Created at Nov 28, 2022 11:01:02

Last updated at Nov 28, 2022 11:01:02

Live data See all activity →

11:01:02 kimutai-de... Create application

End devices (0)

Search Import end devices + Register end device

ID	Name	DevEUI	JoinEUI	Last activity
----	------	--------	---------	---------------

6. After selecting the Register end device, the following screen appears
- Input method – Enter end device specifics manually
- Frequency plan – Europe 863-870 MHz (SF9 for RX2 – recommended)
- LoRaWAN version – LoRaWAN Specification 1.0.1

Register end device

Does your end device have a QR code? Scan it to speed up onboarding.



Scan end device QR code



[Device registration help](#)

End device type

Input Method



Select the end device in the LoRaWAN Device Repository



Enter end device specifics manually

Frequency plan *

Europe 863-870 MHz (SF9 for RX2 - recommended)



LoRaWAN version *

LoRaWAN Specification 1.0.1



Regional Parameters version *

TS001 Technical Specification 1.0.1



7. After that you select the **Show advanced activation, LoRaWAN class and cluster settings** after which the following popup appears
Under the popup select the OTAA option, for additional LoRaWAN class capabilities select class A only , tick the network defaults and under the JoinEUI fill with zeros and press confirm.

[Show advanced activation, LoRaWAN class and cluster settings](#) ^

Activation mode ?

- ☒ Over the air activation (OTAA)
- ☐ Activation by personalization (ABP)
- ☐ Define multicast group (ABP & Multicast)

Additional LoRaWAN class capabilities ?

None (class A only) | v

Network defaults ?

- ☒ Use network's default MAC settings

Cluster settings ?

- ☐ Skip registration on Join Server

Provisioning information

JoinEUI ? *

00 00 00 00 00 00 00 00

Reset

This end device can be registered on the network

8. After confirming the JoinEUI, a pop-up appears where we have to generate the DevEUI and the AppKey.

After generating the respective variables, click **Register end device**

Provisioning information

JoinEUI ? *

00 00 00 00 00 00 00 00

Reset

This end device can be registered on the network

DevEUI ? *

70 B3 D5 7E D0 05 7E E5

Generate

1/50 used

AppKey ? *

78 6C 1E 3C A1 6E 59 93 05 C5 F5 4F 01 6E 38 10

Generate

End device ID ? *

eui-70b3d57ed0057ee5

This value is automatically prefilled using the DevEUI

After registration

- ☒ View registered end device
- ☐ Register another end device of this type

Register end device

9. After that your device is created and the screen below appears where you will be able to copy the APPEUI, DEVEUI and the APPKEY

eui-70b3d57ed0057ee5
ID: eui-70b3d57ed0057ee5

↑ n/a ↓ n/a • No activity yet ☹

[Overview](#) [Live data](#) [Messaging](#) [Location](#) [Payload formatters](#) [Claiming](#) [General settings](#)

General information

End device ID: eui-70b3d57ed0057ee5

Frequency plan: Europe 863-870 MHz (SF9 for RX2 - recomme...)

LoRaWAN version: LoRaWAN Specification 1.0.1

Regional Parameters version: TS001 Technical Specification 1.0.1

Created at: Nov 28, 2022 11:22:32

Activation information

AppEUI: 00 00 00 00 00 00 00 00

DevEUI: 70 B3 D5 7E D0 05 7E E5

AppKey:

Session information

Live data [See all activity →](#)

11:22:32 Create end device

Location [Change location settings →](#)

No location information available

NB – The AppKey should be in big endian format while the AppEUI and DevEUI have to be in little endian format.

- Big-endian is an order in which the Most Significant Bit (msb) is stored first and Little-endian is an order in which the Least Significant bit (lsb) is stored first
- This is easily achieved as shown below

Activation information

AppEUI

0x00, 0x00, 0x00, 0x00, 0x00, 0... lsb ↔ <>

DevEUI

0xE5, 0x7E, 0x05, 0xD0, 0x7E, 0... lsb ↔ <>

AppKey

0x78, 0x6C, 0x1E, 0x3C, 0x... msb ↔ <>

IBM LMIC LIBRARIES

- The MCCI LoRaWAN LMIC library uses the following libraries

```
//Include the Libraries to be used for the LoRaWAN Transmission
#include<lmic.h>
#include<SPI.h>
#include<hal/hal.h>
```

- The <lmic.h> is the MCCI LoRaWAN LMIC library itself
- The hal is the Hardware abstraction layer and is a software subsystem for UNIX-like operating systems providing hardware abstraction
- The SPI.h library allows you to communicate with SPI devices - Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances.
- The MCCI LoRaWAN LMIC Library has pin mappings as shown below

```
//Pin Mapping
//The Board used is a WaziDev 1.3 hence the Pin Mapping is as shown below
const lmic_pinmap lmic_pins = {
    .nss = 10, // It is connected to the Micro-controller
    .rxtx = LMIC_UNUSED_PIN, //Unused
    .rst = 4, // It is connected to the Micro-controller reset pin
    .dio = {2,3,LMIC_UNUSED_PIN},
};
```

Reset – rst

- The transceiver has a reset pin that can be used to explicitly reset it
- The LMIC library uses this to ensure the chip is in a consistent state at startup
- The pin must be configured in the pin-mapping as shown above

RXTX

- The transceiver contains two separate antenna connections, one for RX and one for TX. A typical transceiver board contains an antenna switch chip which allows switching a single antenna between these RX and TX connections. Such an antenna switcher can typically be told what position it should be through an input pin often labelled RXTX
- HopeRF boards such as the one we are using seem to have this connection in place hence they do not expose any rxtx pins and the pin can be marked as unused in the pin mapping.

DIO

- The DIO pins on the SX127x can be configured for various functions
- The LMIC library uses them to get instant status information from the transceiver

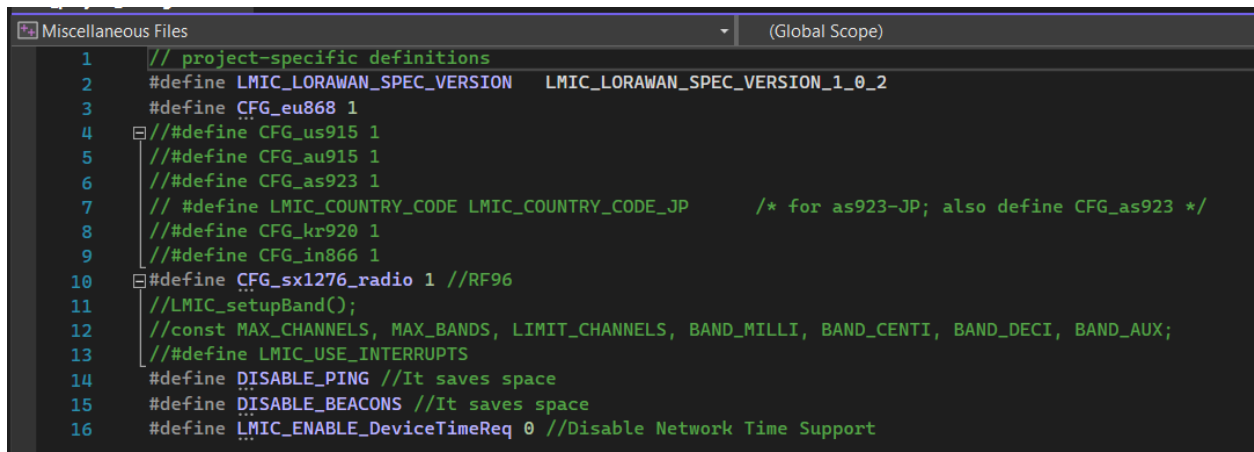
- The LMIC library needs access to DIO0,DIO1 and DIO2, the other DIOx pins can be left disconnected. On the Arduino side they can connect to any I/O pin
- In LoRa mode the DIO pins are used as follows
DIO0 – TxDone and RxDone
DIO1 – Timeout
Hence one pin is left unused as shown in the pin-mapping above

SPI

- The Primary way of communicating with the transceiver is through SPI (Serial Peripheral Interface)
- This uses four pins MOSI, MISO, SCK and SS. The first three need to be directly connected hence for example MOSI to MOSI etc.
- The SS (slave select) connection is a bit more flexible. On the SPI slave side (transceiver) this must be connected to the pin typically labelled NSS. On the SPI master(Arduino) side, this pin can connect to any I/O pin.

NB – Before using the MCCI-LoRaWAN LMIC Library, some adjustments have to be made depending on your location.

- Open the following file location
C:\Users\lenovo\Documents\Arduino\libraries\MCCI_LoRaWAN_LMIC_library\project_config
- Make the following changes



```

1 // project-specific definitions
2 #define LMIC_LORAWAN_SPEC_VERSION LMIC_LORAWAN_SPEC_VERSION_1_0_2
3 #define CFG_eu868 1
4 // #define CFG_us915 1
5 // #define CFG_au915 1
6 // #define CFG_as923 1
7 // #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP /* for as923-JP; also define CFG_as923 */
8 // #define CFG_kr920 1
9 // #define CFG_in866 1
10 #define CFG_sx1276_radio 1 //RF96
11 //LMIC_setupBand();
12 //const MAX_CHANNELS, MAX_BANDS, LIMIT_CHANNELS, BAND_MILLI, BAND_CENTI, BAND_DECI, BAND_AUX;
13 // #define LMIC_USE_INTERRUPTS
14 #define DISABLE_PING //It saves space
15 #define DISABLE_BEACONS //It saves space
16 #define LMIC_ENABLE_DeviceTimeReq 0 //Disable Network Time Support

```

Selecting LoRaWAN Version

- The MCCI LoRaWAN LMIC Library implements V1.0.3 of the LoRaWAN specification however it can also be used with V1.0.2. The only significant change is the US accepted power range in MAC commands
V1.0.2 – 10 to 30dbm
V1.0.3 – 2 to 30dbm
- If none is selected V1.0.3 is automatically selected. In our case
LMIC_LORAWAN_SPEC_VERSION_1_0_2 was selected

Selecting the LoRaWAN region configuration

- The library supports various regions hence in accordance with our frequency selected when registering the end device (Europe 863-870 MHz (SF9 for RX2 – recommended). Our region was selected to be Europe and defined as follows **#define CFG_eu868 1**

Variable	CFG region name	CFG region value	LoRaWAN Regional Spec 1.0.3 Reference	Frequency
-D CFG_eu868	LMIC_REGION_eu868	1	2.2	EU 863-870 MHz ISM

Selecting the target radio receiver

- You should define either one of the following variables. If not, the library assumes sx1276.
#define CFG_sx1272_radio1 – configures the library for use with an sx1272 transceiver
#define CFG_sx1276_radio1 – configures the library for use with an sx1276 transceiver

Controlling use of interrupts

- If defined configures the library to use interrupts for detecting events from the transceiver. If left undefined the library will poll for events from the transceiver.
#define LMIC_USE_INTERRUPTS

Disabling Ping

#define DISABLE_PING

- If defined, removes all code needed for class B downlink during ping slots
- Class A devices such as the one we used don't support ping hence defining #define DISABLE_PING such as the way we did is often a good idea

Disabling Beacons

#define DISABLE_BEACONS

- If defined removes all code needed for handling beacons
- Enabling beacon handling allows tracking of network time and is required if you want to enable downlink during ping slots
- Class A devices don't support tracking beacons, so defining DISABLE_BEACONS might be a good idea

Enabling Network Time Support

#define LMIC_ENABLE_DeviceTimeReq 0

- Disable or enable support for device network-time requests (LoRaWAN MAC requests 0x0D)
- If zero, support is disabled and if non-zero support is enabled

References

<https://www.digikey.com/en/maker/blogs/introduction-to-lora-technology>

<https://docs.aws.amazon.com/iot/latest/developerguide/connect-iot-lorawan-what-is-lorawan.html>

<https://www.semtech.com/lora/what-is-lora>

<https://github.com/mcci-catena/arduino-lmic>