

Building a TTN node

Identifying the components

Our bill of materials (set available at [TinyTronics](#))

- RFM Module RFM95W (868 Mhz)
- Pro Mini 3.3V 8Mhz (5V will NOT work!!)
- Wemos CH340 3.3v-5v TTL USB Serial Port Adapter (has to have a 3.3V option!)
- USB cable for FTDI
- Header 40p divided in: 2x 4p, 1x 6p, 1x 3p and 1x 2p
- PCB 'DougLarue'
- Sensor BME280 (Temperature, Humidity and air pressure), (included in [Sensor Addon](#))
- CCS811 CO2 / tVoC sensor (included in [Sensor Addon](#))
- Battery holder 2x AA
- 2 AA batteries
- 1x female header 2p (battery connection)

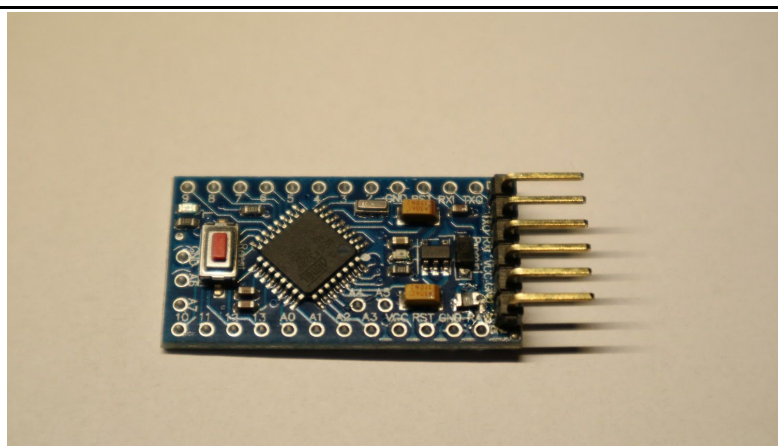
This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Low Power the Arduino?

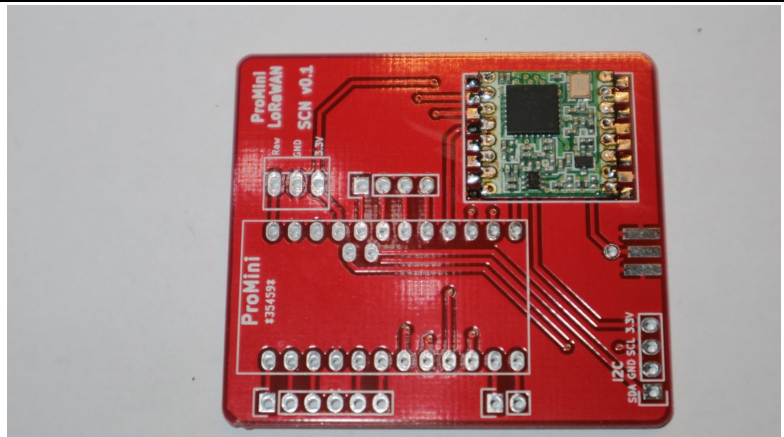
There is an option to 'low power' the Arduino. BUT this requires some precise soldering! The setup will work without removing the components. If you want to low power your node, look at <https://andreasrohner.at/posts/Electronics/How-to-modify-an-Arduino-Pro-Mini-clone-for-low-power-consumption/> for more information.

Building the Node

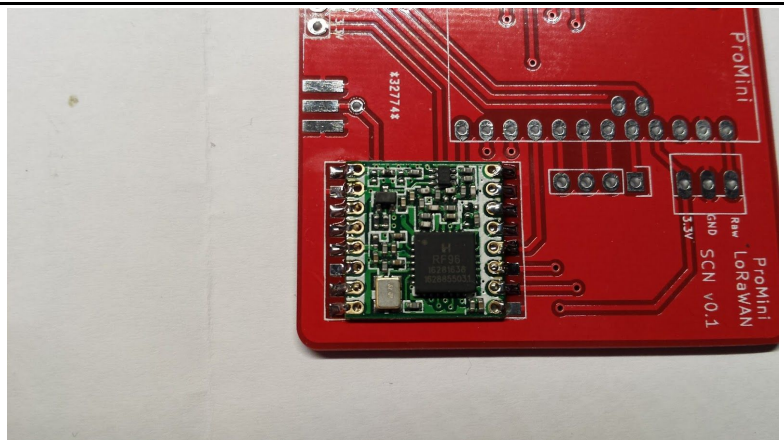
Solder the 6p male header onto the Arduino PCB



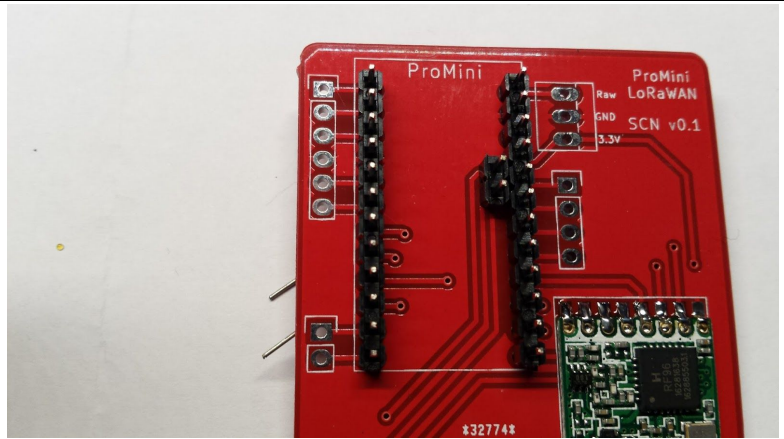
Position the RFM95 radio chip, you might use some Tesla tape to fix it. The crystal should be in the right lower corner (as shown in the picture). Start with just a little solder in two opposite corners.



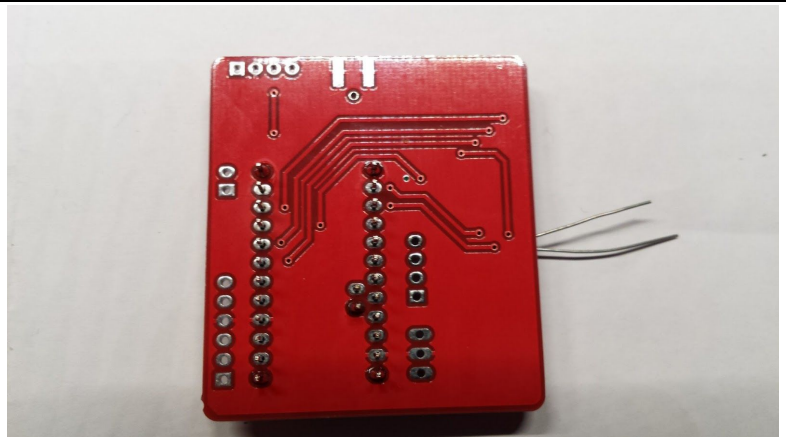
Align the solder pads and if the module is in place, solder the other pads. Do use as little as possible solder! Check the connections carefully!



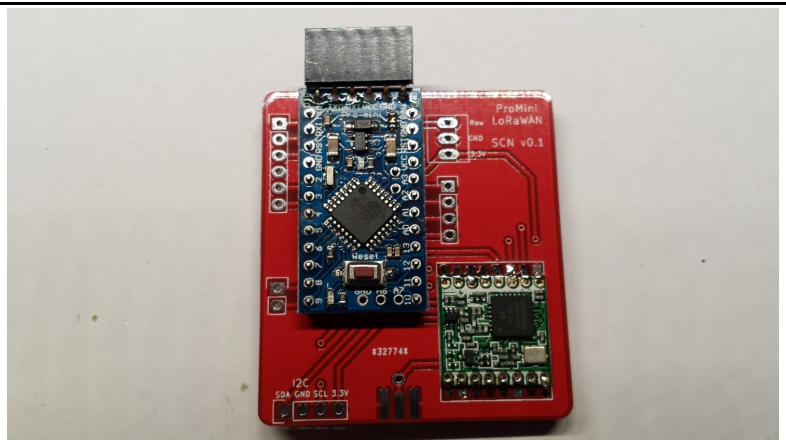
Put two 12p headers into the ProMini socket. Also put the 2p header inside the ProMini area. By holding them with a piece of paper you can turn around the PCB. (You can also use the ProMini to keep the pins in place by using an elastic band.)



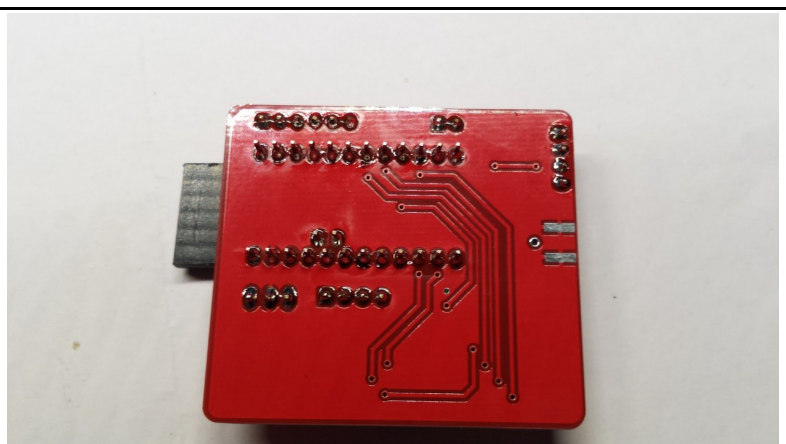
First solder the end pins, align the headers straight and solder all the other pins.

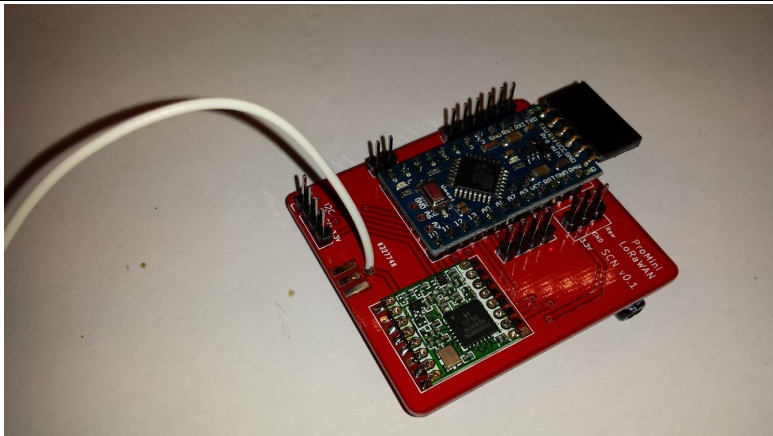
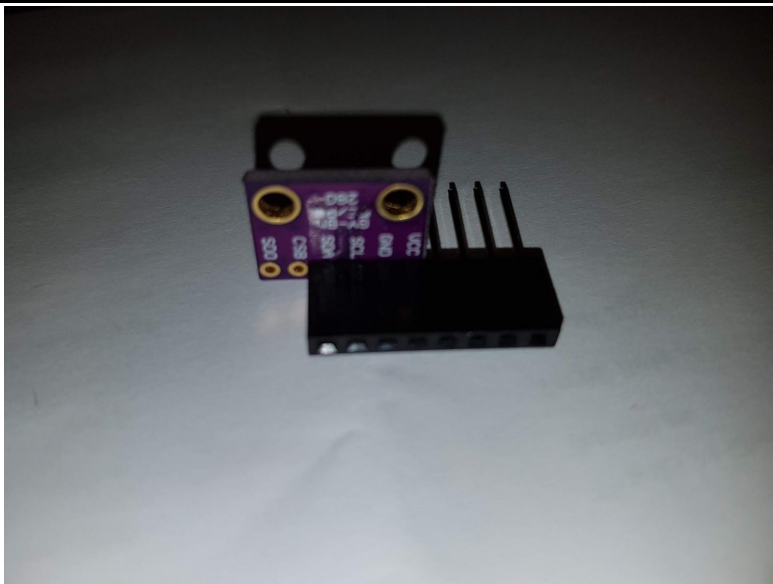


Turn around the PCB and solder the ProMini in place. Do not forget the two pin header!

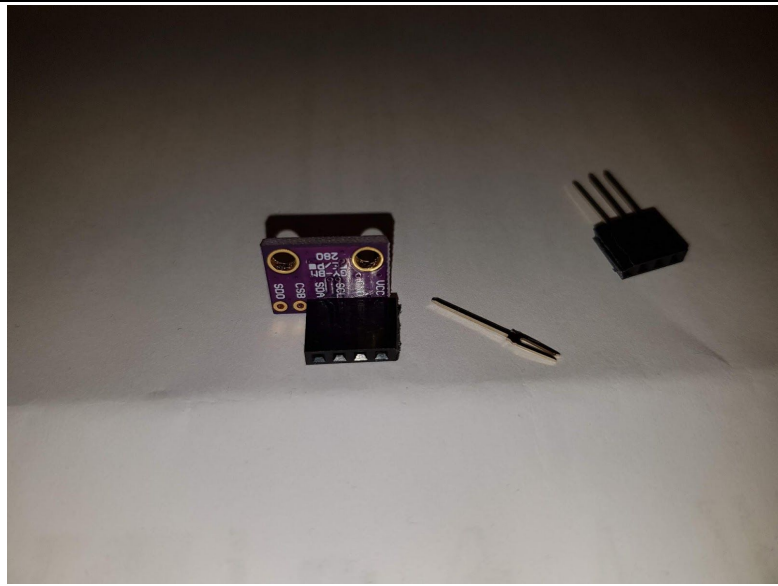


Now solder the 6p, 2p, 3p and 4p header.

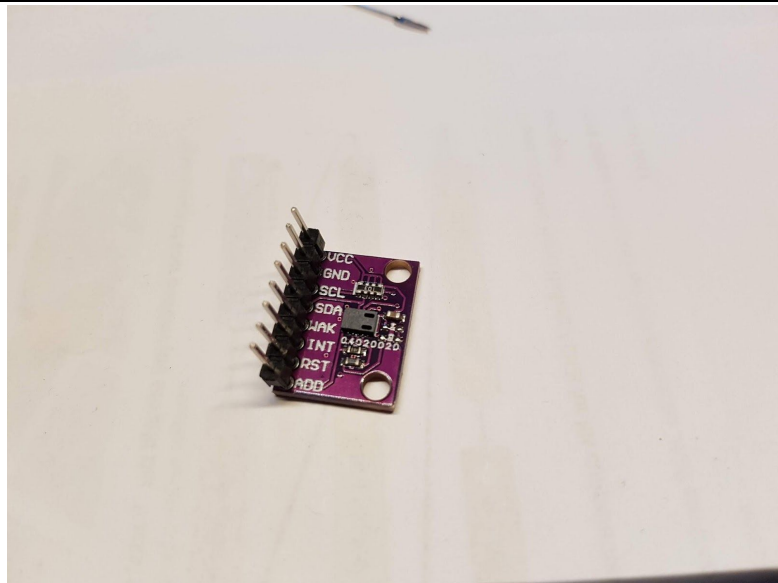


<p>Cut a piece of wire, length 82.2 mm. This is a $\frac{1}{4}$ wave length antenna. You may add some short soldering end.</p>	<p>Wavelength $= 300 / 868 = 0.3456\text{m}$ Wavelength $/ 4 = 0.0864\text{ m}$ correction $0.95 * 0.0864 = 0.0822\text{m}$ (to determine the precise length you can do some experiments, or use proper test equipment like a SWR meter)</p>
<p>Solder the Antenna into the hole near the connector pins. It is also possible to solder a SMA edge connector.</p>	
<p>Solder a header on the BME280 barometric sensor. You may use a 'Arduino Header' so you can use both sides to connect your wires.</p>	

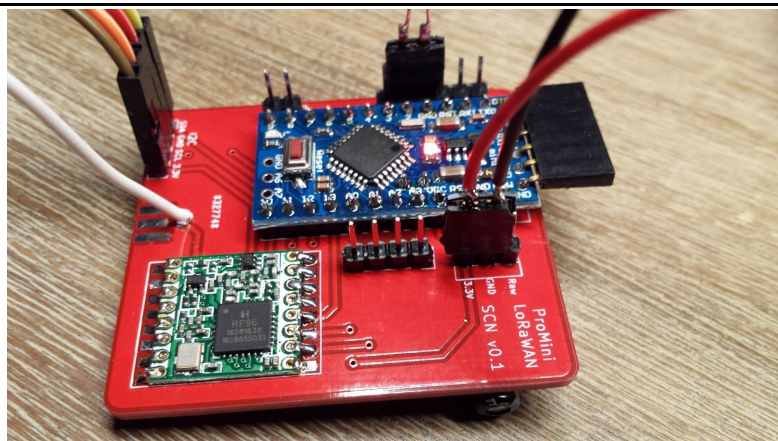
Carefully cut of the remaining pins



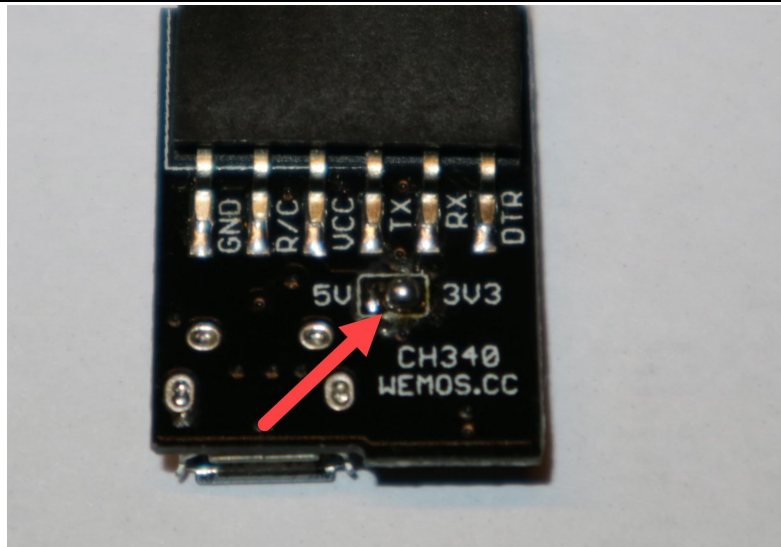
Solder a normal header on the ccs811 sensor



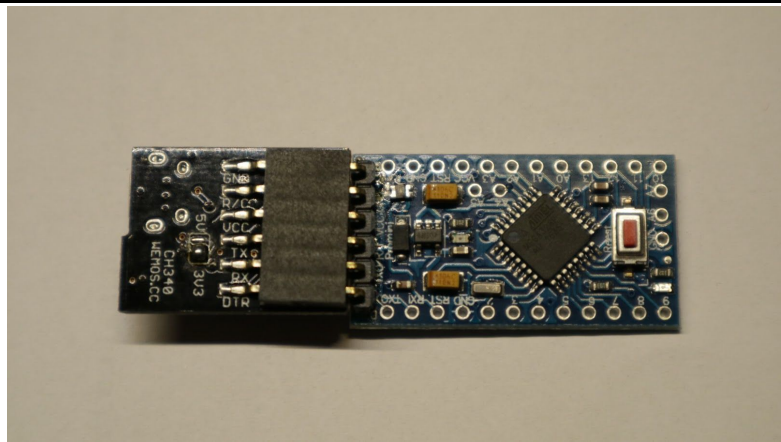
You can solder the battery holder on the pins GND (black) and 3.3V (red), you may use a female header here as well



Ensure the soldering bridge on the FTDI is on the 3v3 position



Connect the FTDI board to the Arduino as shown. DTR should match DTR, GND to GND, Vcc to Vcc, Tx To Rx and Rx to Tx. Check your wiring and connect the FTDI board to your PC.



Starting up the Arduino Environment

<https://www.arduino.cc/en/Guide/HomePage>

Install the Arduino IDE on your favourite platform (Mac, Linux or Windows). These examples are tested with Arduino IDE (tested on 1.8.0)!

<https://www.arduino.cc/en/Guide/ArduinoProMini>

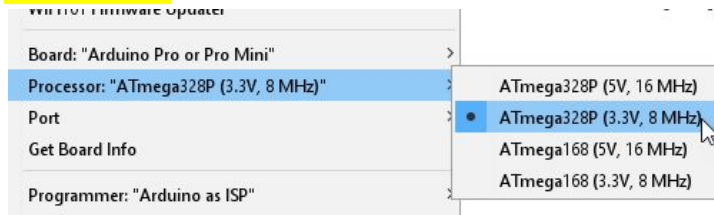
Use the FTDI, ensure that the jumper or bridge is on the '3V' side. Battery power turned off (check the switch).

People using a MAC have a look at this site to update their drivers:

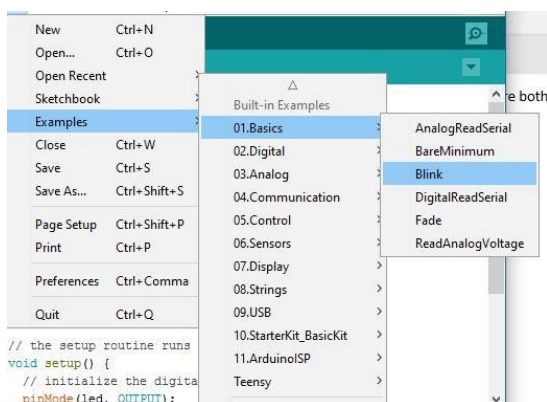
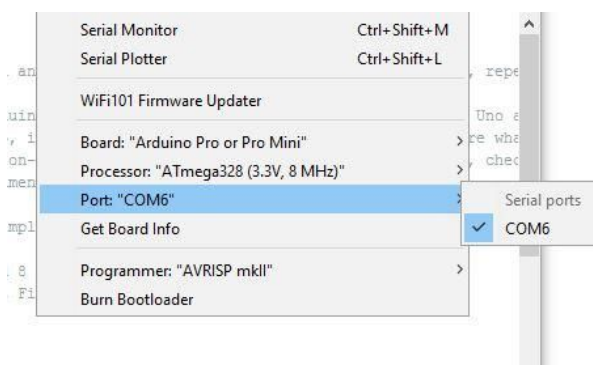
<https://kig.re/2014/12/31/how-to-use-arduino-nano-mini-pro-with-CH340G-on-mac-osx-yosemite.html>

Connect the FTDI and the Pro Mini as shown in the pictures, align the signals: DTR should match DTR, GND to GND, Vcc to Vcc, Tx To Rx and Rx to Tx.

- Start the Arduino IDE
- Select the board: 'Arduino Pro or Pro mini'
- Select the Processor 'ATmega328 (3.3V, 8Mhz) -> IMPORTANT TO SELECT THE RIGHT CLOCK FREQUENCY!!



- Select the COM port given

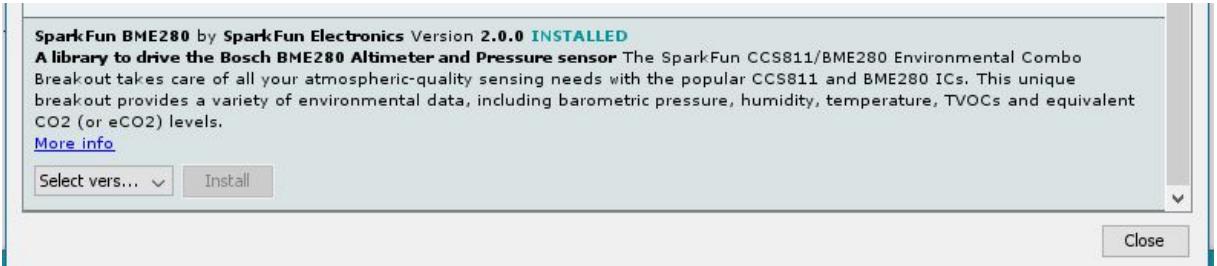


Select the default 'BLINK' example,

Load the code with CTRL-U to the Pro Mini. After compilation and uploading the onboard led should blink at 1 Hz. Great: we have a working programming environment! (Be Aware that pin 13 / onboard LED is used by the RFM95 module as well, so you can NOT use the onboard LED if you want to connect to The Things Network).

Testing the sensors

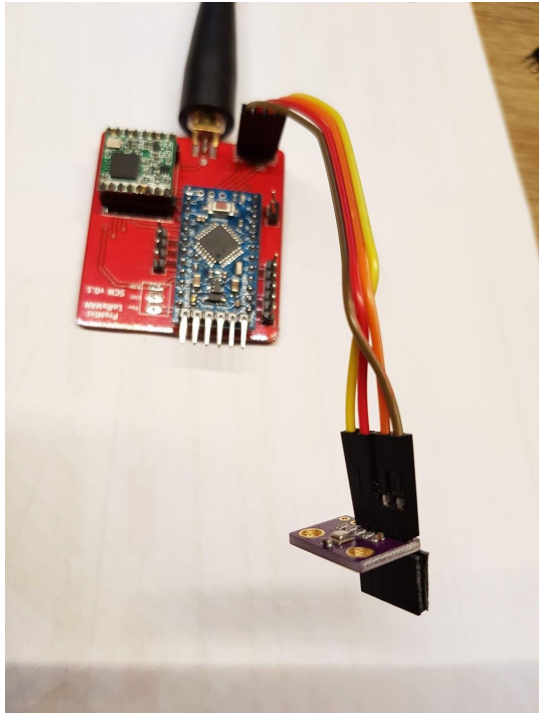
- Add the BME280 library: Sketch-> include library -> manage libraries:
- Search for BME280 and select the Sparkfun BME280 library:

- 

- Search for the Sparkfun CCS811 library and install this one as well:

- 

- Connect the BME280 sensor:

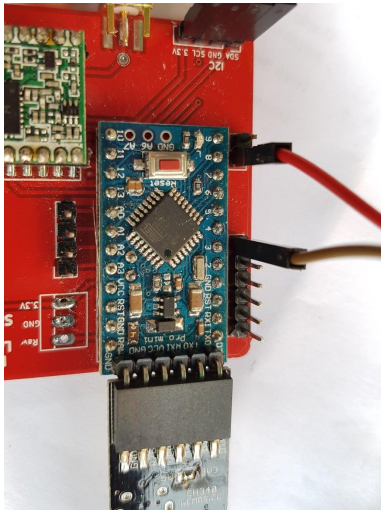


Connect the BME280 sensor with female-female jumper wires to the I2C connector:

- GND-> GND
- SDA->SDA
- SCL->SCL
- VCC->+3.3V

The CSB en SDO are left open (only used in SPI connection). You can not connect straight forward, but you should twist the two wires in the middle. Leave the four 'female' connectors open, we will connect the CCS811 later.

- Now connect the CCS811 sensor. This one is connected to the same I2C bus:



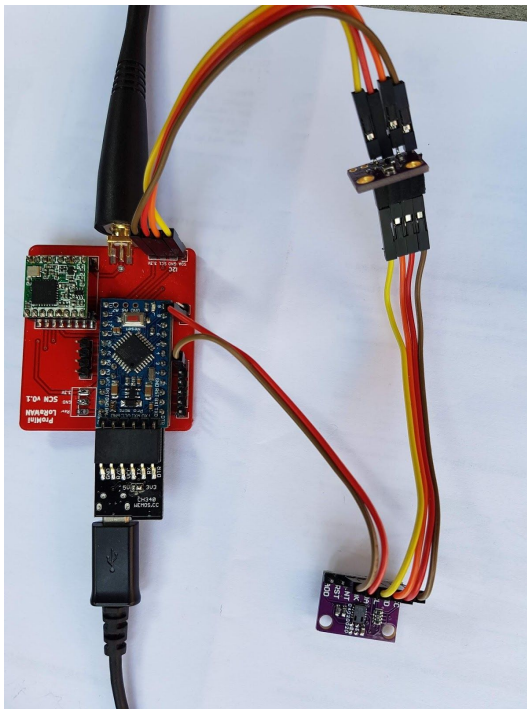
Connect the CC811 sensor with four male-female jumper wires straightthru to the I2C connector:

- GND-> GND
- SDA->SDA
- SCL->SCL
- VCC->VCC

Use two female-female jumper wires to connect WAK and INT to the Main Board:

- WAK -> PIN 8
- INT -> PIN 3

Look at the Arduino pin numbers to select the right pin.



- Get the code from https://github.com/galagaking/wbs_AirQualityNode , wbs_Testing_BME280_CCS811.ino
- Upload the code to your node
- By opening the Serial Monitor (9600 bps) you will see the CO2, tVoC, height, pressure, humidity and temperature of both sensors. Put your finger on the BME sensor (the small metal unit) and see the temperature rise. The sensor's height is relative and this library has a 'fixed' sea level pressure is 1013.25 mBar (you might want to change it at the end of the SparkFunBME280.h file).
- If there are any errors, first check your wiring, if the values still not show up, you may alter the I2C bus addresses as stated in the code, or first start with the wbs_Testing_BME280.ino example.



```
CO2[411] tVOC[1] Humidity[49] Pressure [101269] Alt [4.8] Temp [27.59] Millis [246155]
CO2[411] tVOC[1] Humidity[49] Pressure [101276] Alt [4.2] Temp [27.60] Millis [247191]
CO2[405] tVOC[0] Humidity[49] Pressure [101268] Alt [4.9] Temp [27.61] Millis [248176]
CO2[400] tVOC[0] Humidity[49] Pressure [101277] Alt [3.9] Temp [27.61] Millis [249161]
CO2[408] tVOC[1] Humidity[49] Pressure [101274] Alt [4.4] Temp [27.62] Millis [250146]
CO2[405] tVOC[0] Humidity[48] Pressure [101276] Alt [4.2] Temp [27.62] Millis [251129]
CO2[408] tVOC[1] Humidity[49] Pressure [101277] Alt [4.1] Temp [27.60] Millis [252114]
CO2[411] tVOC[1] Humidity[49] Pressure [101273] Alt [4.5] Temp [27.62] Millis [253100]
CO2[414] tVOC[2] Humidity[49] Pressure [101279] Alt [4.2] Temp [27.62] Millis [254085]
CO2[417] tVOC[2] Humidity[49] Pressure [101276] Alt [4.2] Temp [27.62] Millis [255070]
CO2[427] tVOC[4] Humidity[49] Pressure [101271] Alt [4.7] Temp [27.62] Millis [256055]
CO2[438] tVOC[5] Humidity[49] Pressure [101266] Alt [5.1] Temp [27.64] Millis [257040]
CO2[433] tVOC[5] Humidity[49] Pressure [101269] Alt [4.9] Temp [27.66] Millis [258025]
CO2[425] tVOC[3] Humidity[49] Pressure [101275] Alt [4.3] Temp [27.64] Millis [259010]
CO2[433] tVOC[5] Humidity[49] Pressure [101277] Alt [4.2] Temp [27.64] Millis [259995]
● CO2[448] tVOC[7] Humidity[49] Pressure [101274] Alt [4.2] Temp [27.63] Millis [260978]
```

Getting things started on 'The Things Network'

Download the LMIC library from Github:

1. Goto <https://github.com/matthijskooijman/arduino-lmic>
2. Choose 'Clone or download'
3. Download ZIP
4. Mark the location
5. Go to Arduino IDE
6. Select 'Sketch->Include Library->Add .ZIP Library'
7. Select the downloaded Arduino-lmic-master.zip file from your download location
8. Goto <https://github.com/rockscream/Low-Power>
9. Choose 'Clone or Download'
10. Download ZIP
11. Mark the location
12. Go to Arduino IDE
13. Select 'Sketch->Include Library->Add .ZIP Library'
14. Select the downloaded low-power-master.zip file from your download location

The Things Network Dashboard

Your applications and devices can be managed by The Things Network Dashboard.

Create an Account

To use the dashboard, you need a The Things Network account. You can create an account here:

<https://account.thethingsnetwork.org/users/login> .

After registering and validating your email address, you will be able to log in to The Things Network Dashboard.

Create an Application

<https://console.thethingsnetwork.org/applications>

Choose 'add application'



Give your Application a name, you can ONLY use lowercase! Your description can be any description you like. Choose 'Add application'

A screenshot of the 'ADD APPLICATION' form. It has a title 'ADD APPLICATION' in blue. Below it are four sections: 'Application ID' with a text input containing 'iaq_nodes'; 'Description' with a text input containing 'Indoor Wellbeing Solution'; 'Application EUI' with a text input and a note 'EUI issued by T'; and 'Handler registration' with a text input containing 'ttn-handler-eu'.

ABP

Activation by Personalization (ABP) is a method where the security keys are stored in the device. Not as safe as the OTAA method, but for experiments it works OK. There is no join procedure, nodes will work right away. Choose 'register device':

A screenshot of the 'REGISTER DEVICE' form. It has a title 'REGISTER DEVICE' in blue and a link 'bulk import devices' in the top right. Below it are four sections: 'Device ID' with a text input containing 'iaq_01' and a green checkmark; 'Device EUI' with a text input containing 'this field will be generated' and a pencil icon; 'App Key' with a text input containing 'this field will be generated' and a pencil icon; and 'App EUI' with a text input containing '70 B3 D5 7E D0 00' and a dropdown arrow. An orange arrow points to the pencil icon in the 'Device EUI' section.

Enter a Device ID (lower case), and choose to generate the Device EUI (a pencil will show up).

Choose *Register*.

Now edit the settings (at the upper right of the screen) of the device and choose ABP (OTAA will be

selected by default), and uncheck the frame counter check checkbox

Applications Gateways Supp

iaq_01 > Settings

Overview Data Settings

SETTINGS

Description
A human-readable description of the device

Device EUI
The serial number of your radio module, similar to a MAC address

00 B1 D4 98 D5 8D 8 bytes

Application EUI

70 B3D5 7E D000

Activation Method

OTAA ABP

Device Address

The device address will be assigned by the network server

Network Session Key

Network Session Key will be generated

App Session Key

App Session Key will be generated

Frame Counter Width

16 bit 32 bit

☐ **Frame Counter Checks**
Disabling frame counter checks drastically reduces security and should only be used for development purposes

Delete Device Cancel Save



Select 'save'. Now some values are system generated and we have to copy them to our code. Get the code from

https://github.com/galagaking/wbs_AirQualityNode/blob/master/wbs_BME280_CCS811.ino.

Device Address <> 26 01 1C FE

Network Session Key <> msb { 0xA5, 0x60, 0xFD, 0x60, 0xC1, 0x01,

App Session Key <> msb { 0xCA, 0x23, 0xE9, 0x67, 0xE5, 0x61,

- Copy the Device Address as a HEX value to DEVADDR in the example, so 26 01 1C FE will be 0x26011CFE.
- Copy the Network Session Key as MSB to NWSKEY. (use the '<>' and arrow button to change LSB/MSB and code presentation)
- Copy the App Session Key as MSB to APPSKEY. (use the '<>' and arrow button to change LSB/MSB and code presentation)

```
// LoRaWAN NwkSKey, network session key, AppSKey, application session key, end-device address
static const PROGMEM ul_t NWSKEY[16] = { 0x21, 0xEE, 0x1E, 0x77, 0x58, 0xBD, 0xFE, 0x47, 0x4,
static const PROGMEM ul_t APPSKEY[16] = { 0xFE, 0xC6, 0xE8, 0x6E, 0x4D, 0x31, 0x35, 0x4D, 0xA,
// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x26011CFE; // <-- Change this address for every node!
```

Compile and upload the code. Check in the dashboard the working.

To get the right display format, we can create a decoder in our application. Select your application and open the 'Payload Functions':

PAYLOAD FUNCTIONS

decoder converter validator encoder

```
1 function Decoder(bytes, port) {
2   // Decode an uplink message from a buffer
3   // (array) of bytes to an object of fields.
4   var decoded = {};
5
6   // if (port === 1) decoded.Led = bytes[0];
7
8   return decoded;
9 }
```

Get the payload function from the program code or from Payload.js in Github.

PAYLOAD FORMATS

Payload Format

The payload format sent by your devices

Custom

decoder

converter

validator

encoder

```
1 function Decoder(bytes, port) {
2   var mbar = 970+((bytes[1] >> 2) & 0x3F);
3   var temperature = -2400+6.25*(((bytes[1] & 0x03) << 8) | bytes[0]);
4   var humidity = 5+3*(((bytes[3] >> 3) & 0x1F));
5   var co2 = 4*(((bytes[3] & 0x07) << 8) | bytes[2]);
6   var tvoc = 5*bytes[4];
7   return {
8     pressure: mbar,
9     temperature: temperature / 100.0,
10    humidity: humidity,
11    co2: co2,
12    tvoc: tvoc
13  }
```

Save the function. Return to your data and look what happens:

APPLICATION DATA									
<div> <div> <div>uplink</div> <div>downlink</div> <div>activation</div> <div>ack</div> <div>error</div> </div> <div>Filters</div> </div>									
time	counter	port							
21:40:31	14	1	dev id: iaq_01	payload: 08 A7 95 88 06	co2: 596	humidity: 56	pressure: 1011	temperature: 24.5	tvoc: 36
21:39:28	13	1	dev id: iaq_01	payload: 08 A7 90 88 05	co2: 576	humidity: 56	pressure: 1011	temperature: 24.5	tvoc: 36
21:38:26	12	1	dev id: iaq_01	payload: 08 A7 8E 88 05	co2: 568	humidity: 56	pressure: 1011	temperature: 24.5	tvoc: 36

This way you can put several values into a byte string. Sending in clear ASCII is possible but due to bandwidth limitations not preferable in production environments. To make this even more scalable take a look at <https://github.com/thesolarnomad/lora-serialization>.

Burn in and stabilization

The CCS811 sensor has a resistor component which needs some burn-in time. The onboard processor of the sensor will take care. In normal use it will last approx 10 minutes to get proper results (CO₂>0) and over time the sensor will act more precise after 48 hours 'burn-in' time.

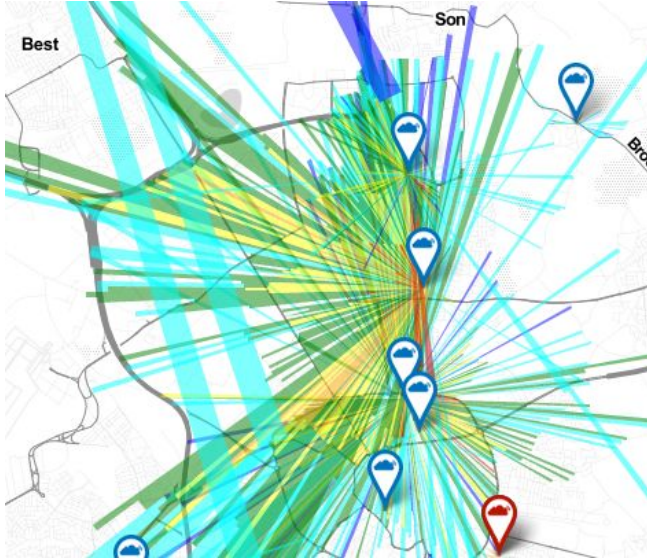
Power Consumption

If you have removed the power LED on the Arduino, the node with the sensors will use 32uA power between the measurements, and 1.3mA during measurement and sending the data (once a minute). This gives approx. a week battery life.



TTN Mapper

On www.ttnmapper.org you can look at the coverage of The Things Network in your neighbourhood.



You can even contribute to this map by using the ttnmapper app on your Android Smartphone. You can use your TTN credentials to select your sensor

LMIC Pin Mapping

If you are using other 'LMIC' or TTN examples, there is ONE part to take care of, the pin setting of the RFM95 module. Most times you have to adjust this to the pinout of the board:

```
// Pin mapping is hardware specific
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = LMIC_UNUSED_PIN,
    .dio = {4, 5, 7}, //DIO0, DIO1 and DIO2 connected
};
```

And even more...

- You can add professional Antenna's on the PCB by using the appropriate connector.
- Support: <https://www.thethingsnetwork.org/forum/>

Thanks to Doug Larue for his PCB design and sharing his manuals.

Frank Beks

June 2018