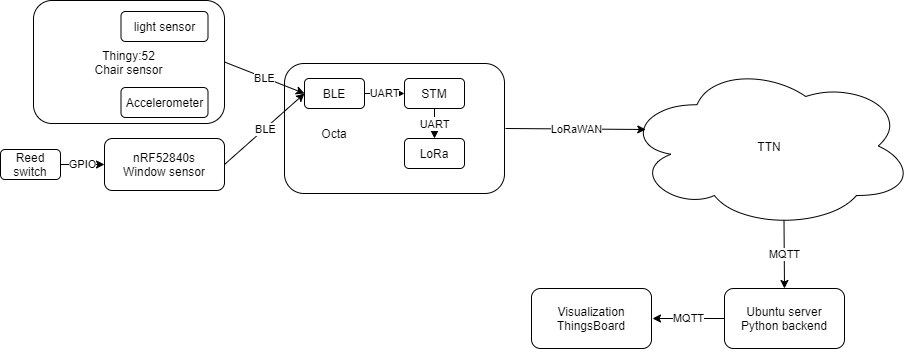
# **Corona Proof Classroom**

This project checks whether the classroom is corona proof by checking whether the windows are open and periodically checking how far people are sitting apart.



## **Window node**

This node informs the system if the window is open or close. As this module should be placed near every window, there is a requirement for an easy mount. This module is entirely wireless to fulfill this requirement. It means that the module communicates via BLE (see subsection Communication), and it is powered independently (see subsection Power harvesting).

This module uses the nRF52840 as a CPU. This CPU was chosen because it has Bluetooth Low Energy (BLE) on-chip. The window status is detected by the reed switch, despite we first discuss using a hall sensor because the hall sensor consumes more power. (For example, the [SS461R.](https://cz.farnell.com/honeywell/ss461r/hall-effect-sensor-bipolar-to/dp/1735770?st=hall%20sensor) consume typically 3.5 mA).

The device should be in power save state most of the time. When someone opens or closes the window, the node should be woked up and report this change. To ensure that the energy supply is sufficient, the device is allowed to send the information only three times per hour. If this limit is reached, it will send the last state at the end of this period. To detect if all devices work properly, the window node will send every 24 hours one message.

### **Power harvesting**

As there could be many Windows node modules in each room, it is essential to make the maintenance as simple as possible. Thus this module harvest energy instead of using batteries, which have to be replaced or recharged after some period.

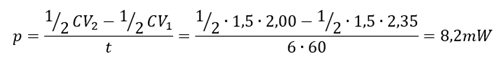
As the window node is placed on the window (as its name says), the best energy to harvest is from the sun. To make the solar panel as small as possible, it is crucial to optimize power consumption.

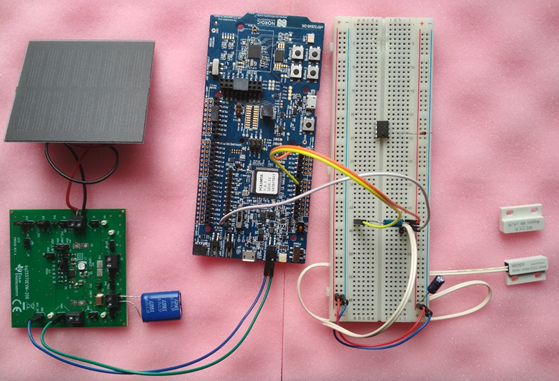
The only source of power we use MIKROE-651 (70.00mm x 65.00mm) solar panel. It could give us 100mA at 4V, meaning that the maximum power is 400mW. As the sensors will be most probably located indoor and can be hidden in the shadow, we do not expect to reach this maximum power.

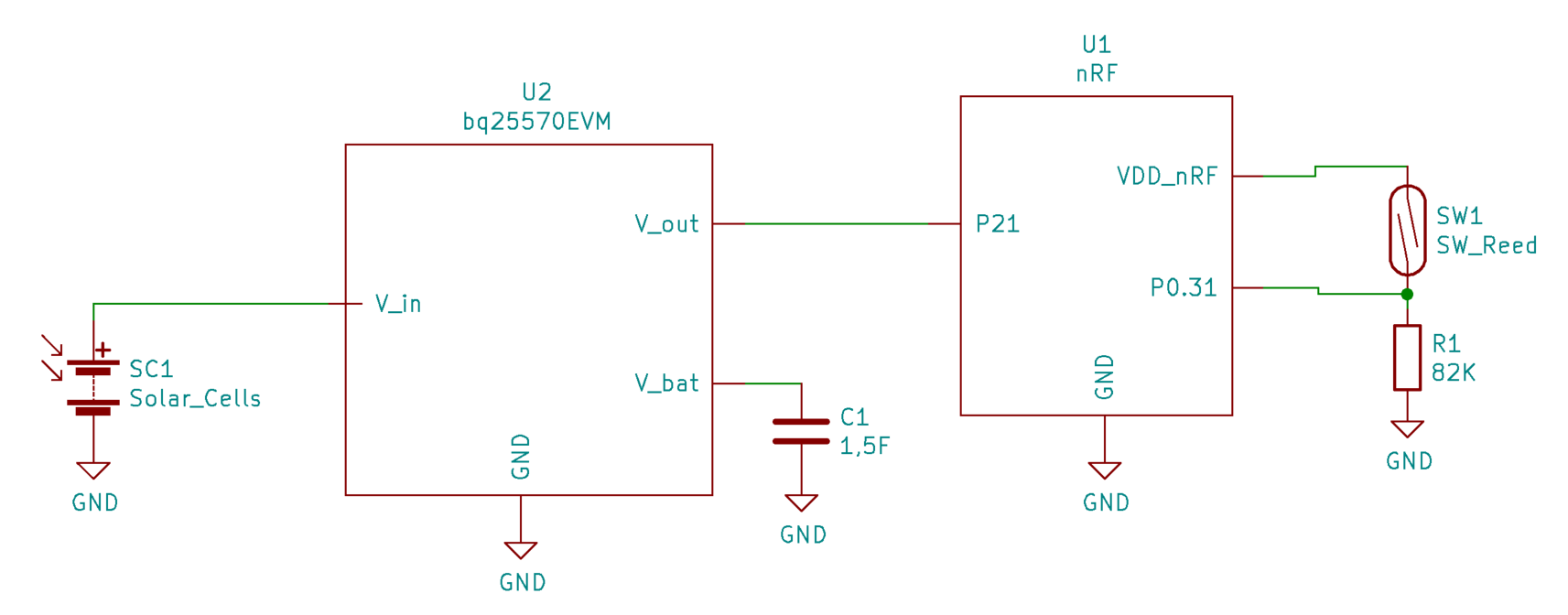
The solar panel is connected to the J2 screw terminal of the [bq25570EVM-206](https://www.ti.com/lit/ug/sluuaa7a/sluuaa7a.pdf?ts=1606856067193&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FBQ25570EVM-206%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dapp-null-null-GPN_EN_EVM-cpc-evm-google-wwe%2526utm_content%253DBQ25570EVM206%2526ds_k%253DBQ25570EVM-206%2526DCM%253Dyes%2526gclid%253DCjwKCAiA8Jf-BRB-EiwAWDtEGso9PlQg4hH1DbfHmVWag5CSS1KGx1ErZqPrdqqDqYDGDrUgbuuwxhoCmHsQAvD_BwE%2526gclsrc%253Daw.ds) evaluation board, which is well-suited for high impedance power direct current sources. The maximum output power point(MPP) of those power sources varies with ambient conditions. In our case, the solar panel's MPP varies with the amount of light and the temperature. The bq25570 automatically track the MPP to maximize the amount of harvested power. We set the MPP is located at 80% of open-circuit voltage by connecting JP4 to 80%. Only one of JP4 and JP1 can have a shunt installed at the same time.

This power management module charges a supercapacitor or Li-pol battery, which ensures the availability of power. This storage element can have significant power leakage. We decide to use the supercapacitor 1.5F instead of a battery, as it is less expensive. The supercapacitor is connected to J8. The power management module can charge the supercapacitor up to 4.2V. On our development board, the maximum is set to 3.2V via resistor divider (R1 and R2). I measure maximum voltage 2,4V. There is internally programmed undervoltage protection of 2.0 V to protect the Li-pol battery. Once the battery voltage is above undervoltage level, it provides 1.8 V to its output (J11). This voltage is set via resistors R9 and R10. The output is enabled when a shunt is installed on JP2 (connect \EN and GND). The output bug converter can be disabled using JP3 and JP6. We connect VOUT\_EN and VSTOR on JP3 and discontent JP6. In this configuration, the device is always powered on when the storage element is not undervolted.

We measured the power of the solar panel in a real environment. The whole setup was connected, as described earlier. The J11 was unconnected. We measured the voltage on the supercapacitor (V1). After six minute we measure it again (V2). Then we can compute the power as follow:







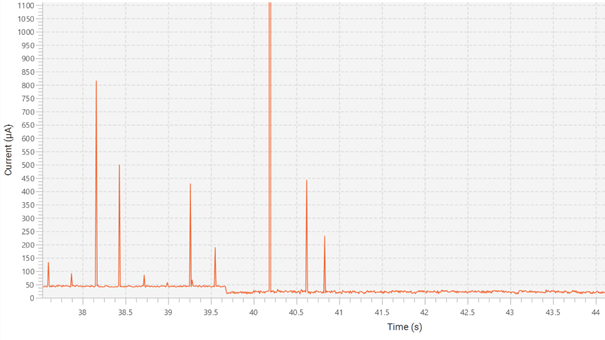
### **Power savings**

The power consumption was measured using STM X-NUCLEO-LPM01A. For the first measurement, the value from the screen was used because it is a single value, and it is easy to compare. For the final measurements, the current was the current-time profile was captured using STM32 CubeMonitor-Power.

For the first measurement device was connected to V\_in 3-5 pin on P20. In this scenario, the board consumes approximately 5mA. When changing the SW6 to "nRF only," the power consumption drop to 2.095 mA when the magnet is present to the reed switch and 2.408 mA when not present. This significant difference is caused by the pull-up resistor. In this scenario, the internal pull-up was used. The best way to solve this problem is to use a different reed switch, which will switch between power and ground (For example, [DRS-DTH-50-80](https://cz.mouser.com/datasheet/2/240/littelfuse_reed_switches_DRS_DTH_datasheet_pdf-1372468.pdf)), so there will be no need for a pull-up resistor. The second possible solution, which we used, is to use an external pull up with higher resistance. We use R = 180 kΩ, which causes a 17 µA leak when the power voltage is 3 V and 10 µA when the power supply goes to 1,8 V. This may increase the risk of radio-frequency interference. For this reason, it important to have the wire as short as possible.

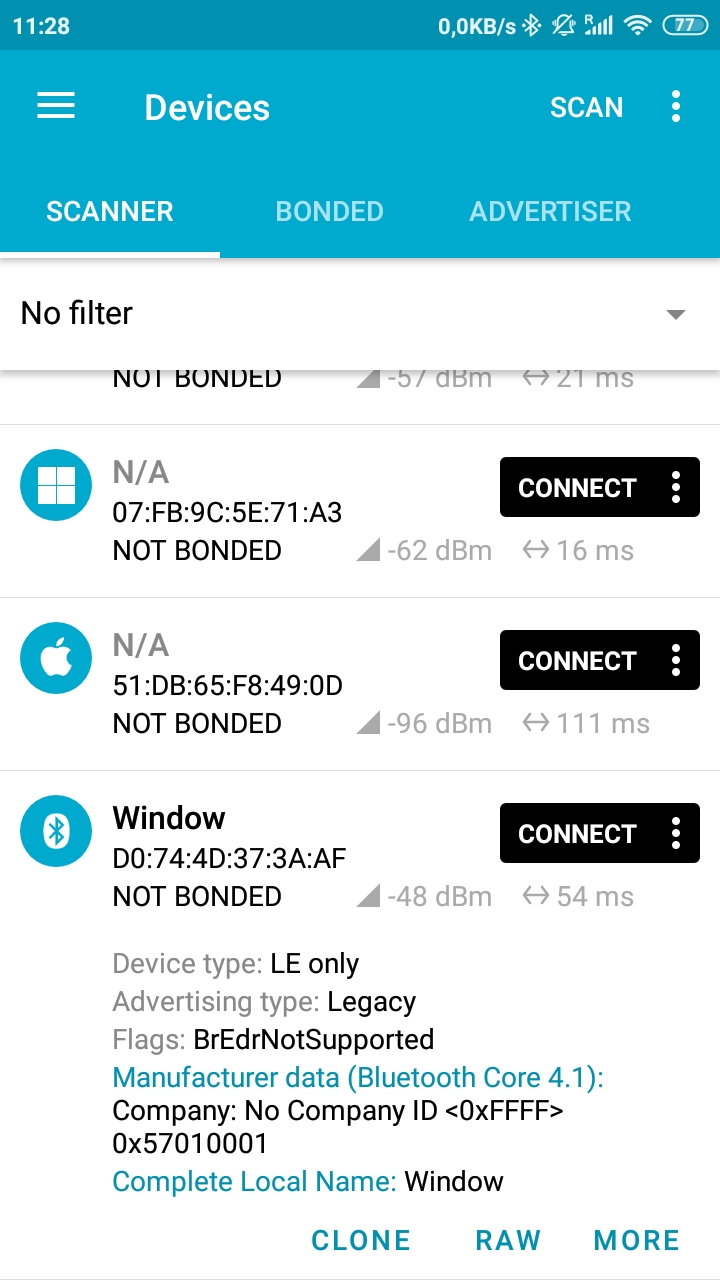
A second significant power consumption drop was done by using P21 to power it up. In this scenario, the current drops to 50 µA. Disable the UART logging to save some more power and proper initialization of timers. Rather than using the wait function, we managed to decrease the power consumption to 20 µA when there is a magnet and 40 µA without the magnet. As you can see, there is a current loss of 20 µA when a magnet is present, from one half the resistor causes it, and the second half can be caused by the current going into the PIN (gate of the MOS structure and protection diodes).

The result is shown in this picture. Firstly, there was a magnet present (the window is closed), and after some time, the window is open. The device waits for 0,5s (debouncing) and then start the advertisement. The advertisement is the biggest peak. The whole system is powered by 1,8 V.



### **Communication**

The window node communicate with the OCTA using BLE advertisement. It displays the complete local name, which is Window and all useful information is in the manufacturer data. There is no company (0xFFFF); ascii code of W(0x75); than 8-bit long ID; zero byte (to make it the same length as chair node) and 1 if window is closed or 0 if it is open.



## **Octa**

The Octa is in charge of receiving the BLE advertisement messages, parsing the data and sending newly received data to TheThingsNetwork using LoRaWAN. The nrf52832 chip on the Octa constantly checks for advertisement messages. When an advertisement from a device with a complete local name starting with “Thingy” or “Window” gets detected, the advertisement will get parsed. We had to change some code in the ble\_advdata.c file, which is part of the NRF SDK, to make the filter scan on part of the name and not the whole name. The parsed data is sent to the Nucleo board via UART, using an interrupt, for further processing. The interrupt fills a buffer that has a length of 6 bytes. Our data is 4 bytes long and because our custom data is stored in the manufacturer data, we also get 2 bytes manufacturer ID.

A thread is always running in the background that checks the buffer and compares it to an array containing the received messages. This array has a length that corresponds to the number of sensors used. This also means that the array only stores the most recent messages. A new message gets added to the array. A new message coming from a sensor that has sent a message before, will replace the old message. Finally messages containing the same information from the same sensor will get ignored, these messages only indicate that a sensor is still alive and the state has not changed. If a new message is received, a recentData flag is set. This flag is needed so the Octa only sends when it has new data and thus does not, for example, send at night. To send data to TTN, the LoRaWAN\_send function is used. It is called every 90 seconds using a timer. We are using spreading factor 7 at 125 KHz and according to the Fair Use Policy of TTN we can only send 20 messages per hour. We can make this 40 messages per hour by not sending at night. For our project we need to send as many messages as possible to detect in time when people are sitting too close to each other and therefore we turned off ADR and force the chip to use spreading factor 7. We only use 3 sensors for the demo, that is also one of the reasons we can send upto 40 messages per hour. If we were to use more sensors, the payload would increase, therefore the airtime would increase and we would be able to send less messages.

### **Power**

The Octa is allowed to be powered by battery. The battery should be about 5885 mAh according to following calculations:

* STML4:
  + 31 µA/MHz ⇒ Run at max speed (80 MHz): 31µA \* 80 = 2.48mA ⇒ 2.48 mAh
* nRF52832:
  + Rx @ 1Mbps: 5.4 mA ⇒ 5.4 mAh
  + No sending
* CMWX1ZZABZ-091 (Murata)
  + Transmitting @ 14 dBm (default): 47mA
  + 40 messages per hour
  + SPF 7 / 125 kHZ, max payload = 222 bytes
  + Our payload = 4 bytes. For 3 sensors, max payload = 12 bytes
  + 61,7 ms airtime with these specifications
  + Total: 2.48 mAh + 5.4 mAh + 0.03 mAh = 7.91 mAh
  + 31 days = 744 hours ⇒ 744 \* 7.91 mAh = 5885 mAh

## **Backend**

The python backend uses MQTT to receive data from TTN and uses MQTT to send/publish data to the ThingsBoard. On top of that, it also stores the data in a MongoDB database. Currently it only stores the most recent data from a sensor and replaces old values. This can be changed by, for example, adding a timestamp. When the backend gets data via MQTT from TTN, it parses the complete message. One LoRaWAN message can contain messages from multiple sensors and so we need to split this message and parse them separately. The right number of Window and Chair objects are created accordingly. These objects are then used for 2 things. Firstly, the MongoDB database gets updated with these new values. Secondly, a new message is created to publish via MQTT to inform the ThingsBoard of the incoming changes.

**Thingy**

See “Portfolio\_Thingy\_Arne\_Defays” and “Power Consumption estimation” in github repo => when I wanted to copy paste everything to here, it didn’t work.

## **Visualization**

We use the<https://thingsboard.cloud/> to visualize the data. You can log as a user using a demo customer account:

Username: corona.proof.classroom.user1@mailinator.com

Password: EasyDictionaryPassword123!

When logged in, choose Dashboards, then CoronaProofClassroom, and Open entity group. Now you can choose a campus. You can pick it up from the table or the map. For each campus, there is a plan. Here you can select the building both from the plan and the tabular. Then you can choose the room. For testing purposes, there is a "Virtual room". Data in this room is automatically generated. In the room, you can see the layout. There are windows icons in the layout, which are changing accordingly to the real world. There is a table where is listed all connected chairs sensors and their neighbor ID + RSSI. The RSSI indicates the distance between chairs.

