Lucie Bechtet
Isabella Mi Hyun Kim
Philippe Hérail

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Microcontroller on open source hardware report

I - The Arduino part

A. LoRa module

In order to interface the LoRa module with the Arduino board, we first soldered the board on a PCB, to which we added male header pins for a simple usage with a breadboard. Then, to simplify the usage, we used a library, which wraps some of the most used functions (connect to The Things Network (TTN), send a message). This allows us to only use the text commands (listed on the module datasheet) only to change specific parameters.

Using this library, we were able to send messages to the INSA gateway, as well as receiving acknowledgements from it.

B. Gas Sensor

Once our LoRa communication was working, we tried to interface a sensor with our board, in order to build a more realistic application. We used the a gas sensor, which was readily available during the practice sessions.
We used the embedded arduino ADC to convert the data from the sensor, which we then sent to the LoRa gateway. The first iteration of the application periodically scanned polled the sensor and sent its data.

Unfortunately, this was not power efficient: indeed, if we imagine a scenario where the sensor is used to detect dangerous levels of a given gas, we only need it to send an alert when a threshold is reached. Thus, we may want to have the board in sleep mode when it has nothing to do. Our first attempt at this used the integrated analog voltage comparator (ACMP), which was able to trigger an interrupt when the sensor reached a set voltage. Unfortunately, when we wanted to set up the power saving mode, we could not use this ACMP: indeed, the datasheet specifies that it may not be used when the MCU is in sleep mode, only GPIO level change interrupt may be used in our case.

In order to use the these GPIO interrupts, we had to have a way to trigger a level change when the sensor output level reached a set threshold. Thus, we use a schmitt trigger, in order to detect a level change, while automatically debouncing the signal thanks to the intrinsic hysteresis cycle.

ADD SCHEMATIC + VALUES

Once this trigger was set, we setup a rising edge interrupt in order to detect only when the sensor reaches the level, not when we leave it. One may argue that both detection makes sense for our application, as we may want to detect when the dangerous gas alert can be called off, however we chose to keep our application simple in this case.

C. Power consumption evaluation & reduction

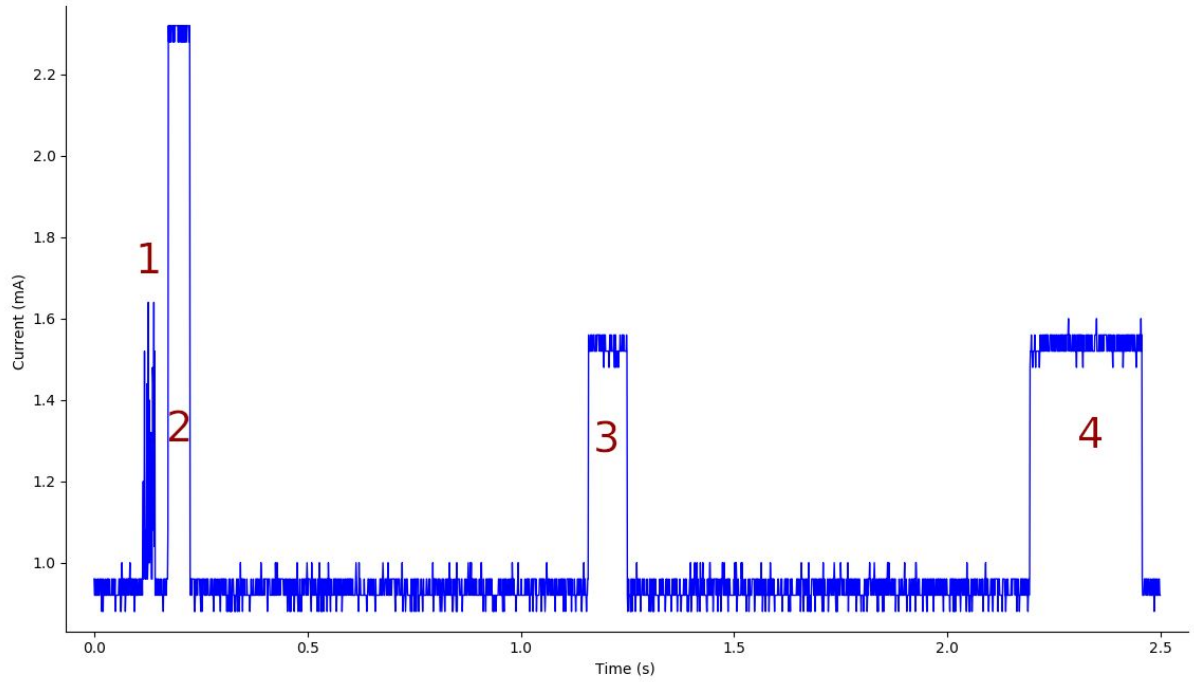
One of the main power consumer of our system is the Arduino board. Therefore, as said in the previous part, the MCU spends most of its time not doing anything. Therefore, we put it to sleep at the end of each iteration of the loop using this function:

```
void putBoardSleep()
{
   sleep_enable();
   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
   sleep_cpu();
}
```

This allows us to put the board to sleep, and resume (which makes the function return) only on interrupt.

We used a USB power detector to measure the consumption of our full system. Unfortunately, this was not a very precise measurement mean, and the power consumption was shown to be similar with or without the sleep mode. As there are a number of leds on the board, our assumption is that they incur most of the power consumption on the system. As we had no way to measure the current supplied only to the MCU, without the embedded LEDs (on the shield, on the USB TX/RX pins...), we cannot confirm this assumption.

However, we are able to measure the current consumed by the LoRa module, in order to evaluate its participation in the power consumption. To this end, we used a shunt resistor in series with its input voltage.
At first we used a 1 kΩ resistor to measure the power consumption through the voltage drop using an oscilloscope. This resulted in a ~1V drop, which is a lot considering we powered the module with 5V. This did not prevent it from working properly, therefore we wanted to use 10 Ω resistor, to incur a drop between 10 mV and 20 mV, but this small value required very fine settings of the oscilloscope, and was therefore abandoned in the time we had left.
The value obtained in the first run are shown below:



Current consumption in function of time

These readings are highly interesting, not only because we can precisely evaluate the power consumption of the module, but also because we can observe the different phases of the LoRa communication. These phases are defined below, in correspondence with the numbers on the picture, along with the energy they consumed:

1. Message sent to the module (85.1 µJ): this part is not LoRa in itself, it corresponds to the message sent from the MCU to the module and its processing before transmission.
2. Message transmission (359 µJ): this is the message transmission part of the LoRa protocol.
3. Reception window 1 (409 µJ): this is the first RX event from the LoRa protocol
4. Reception window 2 (1.31 mJ): this is the second RX event from the LoRa protocol. Being much longer than the first one, we can assume that the acknowledgement message was detected and received during this event.

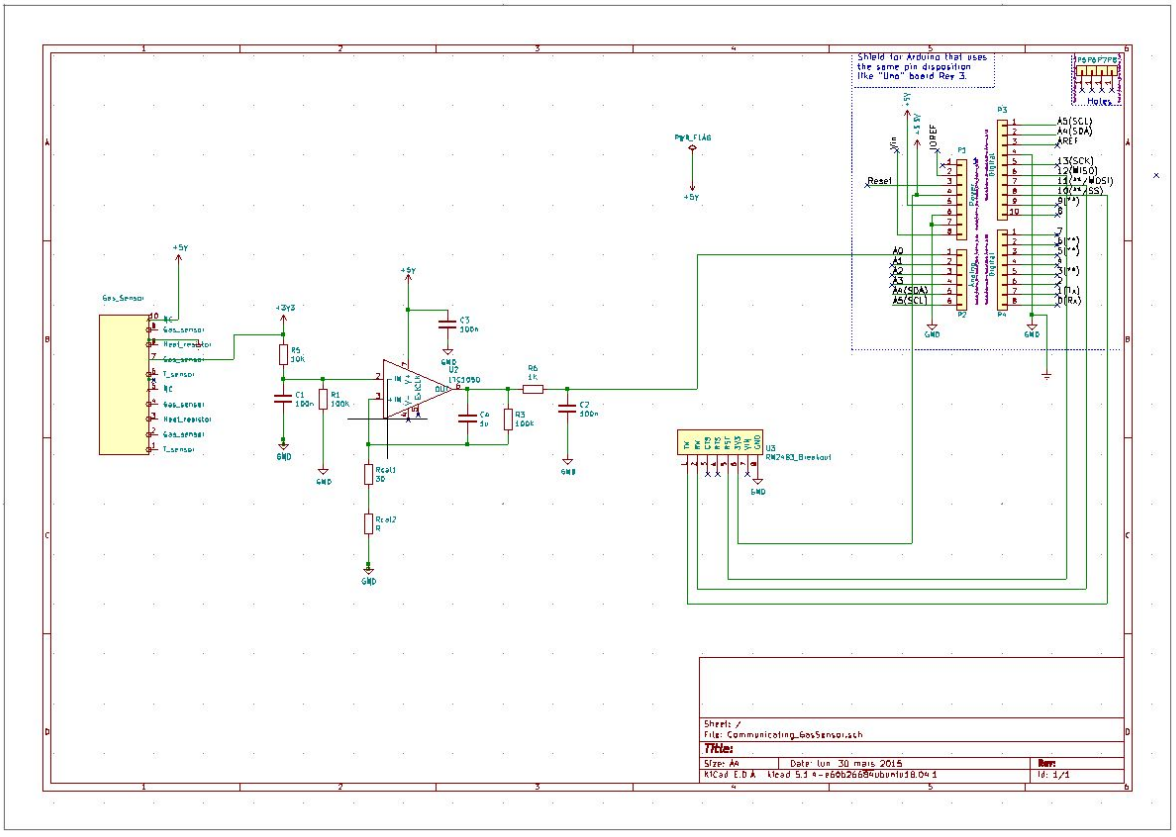
Therefore, the total energy consumed by the LoRa module turing transmission, from the MCU message to the end of the reception event is 8.61 mJ, and the power consumed in standby mode is 3 mW.

II - The Kicad part

The PCB designed in the kicad files includes the arduino shield, the gas sensor, the operational amplifier LT1050 to adapt the gas sensor values as inputs to the board, and the Lora module RN2483.

The Arduino shield is already present in a library with its template, but we designed manually the module and the sensor. We also designed their footprints.
After the electrical schemes of each components are done, we have to link them and then to run the errors finding tools of Kicad. We had some issues at this point because of several things we did not know on the Kicad software.

We had some trouble to create the connections because sometimes they seemed to be done to the naked eye but were actually just drawn and did not really connecting two pins. Even zooming on the pins to connect, we had to try several times to create proper connection.
Another trouble we had is that we didn't know that the unconnected pins had to be marked with a cross. This was a minor issue easy to solve.
We however encountered problem for the power supplies. We had to connect the +5V to a power flag that is invisible on the electrical schemes, and we needed time to understand the issue raised by the "error solver".



After solving all the electrical issues, we associated each element to a footprint. For the module and the gas sensor, we associated it to the footprint we created ourselves. The other components were associated to footprints already presented inside of the libraries.

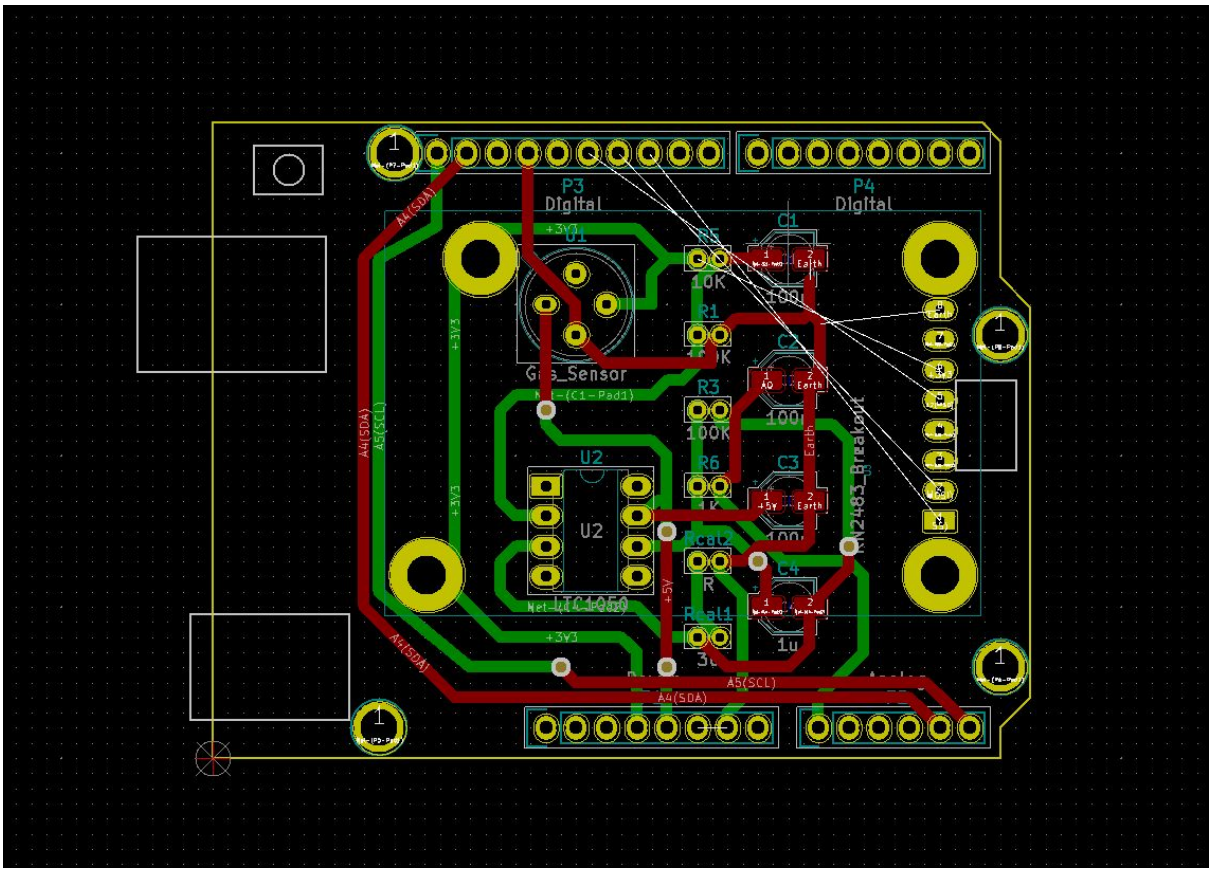
Symbol : Footprint Assignments

1	C1	100n	Capacitor_SMD:CP_Elec_4x3
2	C2	100n	Capacitor_SMD:CP_Elec_4x3
3	C3	100n	Capacitor_SMD:CP_Elec_4x3
4	C4	1u	Capacitor_SMD:CP_Elec_4x3
5	P1	Power	Socket_Arduino_Uno:Socket_Strip_Arduino_1x08
6	P2	Analog	Socket_Arduino_Uno:Socket_Strip_Arduino_1x06
7	P3	Digital	Socket_Arduino_Uno:Socket_Strip_Arduino_1x10
8	P4	Digital	Socket_Arduino_Uno:Socket_Strip_Arduino_1x08
9	P5	CONN_01X01	Socket_Arduino_Uno:Arduino_1pin
10	P6	CONN_01X01	Socket_Arduino_Uno:Arduino_1pin
11	P7	CONN_01X01	Socket_Arduino_Uno:Arduino_1pin
12	P8	CONN_01X01	Socket_Arduino_Uno:Arduino_1pin
13	R1	100K	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
14	R3	100K	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
15	R5	10K	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
16	R6	1K	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
17	Rcal1	30	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
18	Rcal2	R	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P1.90mm_Vertical
19	U1	Gas_Sensor	Lib_GasSensor:TO-5-4
20	U2	LTC1050	Package_DIP:DIP-8_W7.62mm_LongPads
21	U3	RN2483_Breakout	Lib_GasSensor:RN2483_Breakout

Once the footprint are associated and the electrical schemes are done, we can pass to the PCB interface. To make the PCB connections, we placed all of our elements in our board in order to distribute all of our components equally in the board and at the same time considering the placement of the shield above the board.

To make the connections we used the Front Copper layer (in red) and the Back copper layer (in green). That mean that these connections are made in opposite sides of the board. In our case, we used SMD capacitors that use surface mount technology, so they had to be connected using the Front Copper layer.

For all the other components. we could use the either the Front or the Back copper layers. And alternate between these two layers to avoid cross overs.



After all the PCB connections done, we could visualize in 3D our board.