Smart beer coaster

Alessandro Luchetti Mat. 180061
Cristiano Strobbe Mat. 196763
Marco Basilici Mat. 182716

Davide Brunelli
Maurizio Rossi

January 29, 2018

Abstract

This document aimed to be a report of a project for the Embedded Systems master course at the University of Trento. The main scope of this project is to develop a compact device able to detect the presence or absence of beer inside a glass and to communicate it to a User Interface. Thanks to this implementation many waiters can be helped in their work.

I. Introduction

A first market survey was made to find if there are other products with the characteristics that we want to implement in our device. They can be summarized in the following ones:

- Low consumptions
- Device compact as possible
- Able to read beers level and also able to call the waiter
- Radio communication
- Rechargeable
- Waterproof
- Cheap
- Easy to use

Initially our idea was to implement these characteristics inside a glass but since this device wants to be something salable for a factor cost it seemed more appropriate to make it as an accessory, like a coaster, to be introduced in the bars rather than replace all the glasses.
The main problem was to find a good way to measure liquids level. After long searches we decided to measure the liquids level indirectly by measuring the pressure applied by the glass on the coaster.
In this report we will explain how we implemented this way.
The project can be divided into six macro areas which are describe in this paper as follow:

- Sensors
- Software
- LoRaWAN
- Power supply
- TheThingsNetwork
- Design

The electronic and the software tools used to develop, design and realize this device will be mentioned in this paper together with their results obtained.

II. Overall scheme

The figure 1 shows the overall configuration of our project.

The brain of our project is the new *STM32L072CZY6TR* micro-controller. This one is used because of its ultra-low-power consumptions and because it integrates both a micro-controller and both a RF communication module. The layout of the micro-controller is shown in figure 2. While the discovery kit that we used for our tests is shown in figure 3 .

III. Sensors

i. Force Sensitive Resistor

Since we decided to use a coaster we needed a sensor that allowed us to detect physical pressure or weight. The most promising sensors

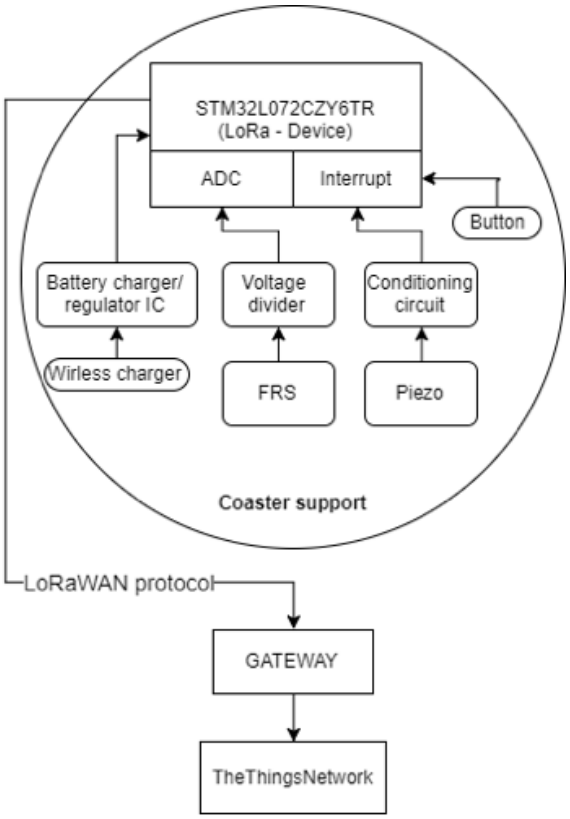

Figure 1: *Functional scheme*

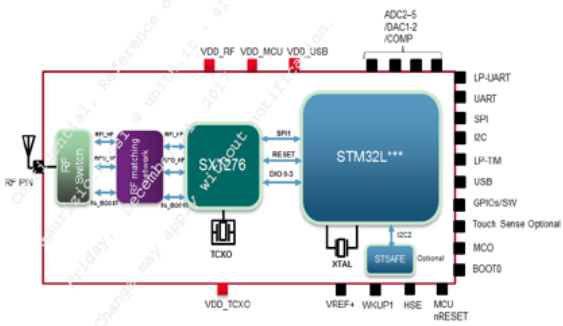

Figure 2: *STM32L072CZY6TR layout*

that we found were: strain gauges (load cells integrated inside the coaster) and Force Sensitive Resistors (*FSR*). We choose *FSR* because the output signal does not need a conditioning and it's simpler to mount (no special glue needed).
The sensor is basically a variable resistor which is composed by two layers of a conductive material separated by a third one which is non-conductive, figure 4. The higher will be the force/pressure that is applied the lower will be the resistance. Typically, the resistance range varies from 1 MΩ (*unload condition*) to some ohms (*maximum load condition*). The figure 5 shows the transfer function between the input force and the output resistance (linear in logarithmic scale). As we can see from the figure 5 the slope of the transfer function increases for small weights. This means that when a little force is applied then the resistance goes quickly from ~1 MΩ to some tens of KΩ.
The drawback of this sensor is the fact that the output depends on how homogeneously

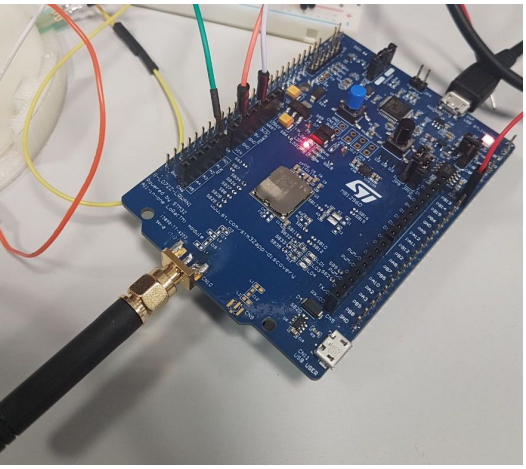

Figure 3: *STM32L0 Discovery kit LoRa*

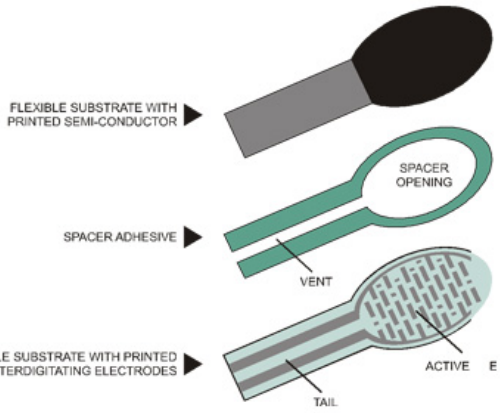

Figure 4: *FSR layer stack*

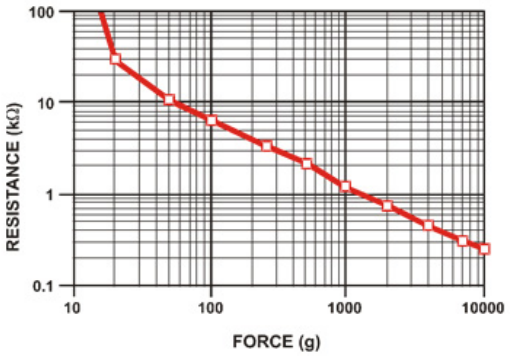

Figure 5: *FSR tranfer function*

the force is applied. We analyzed two sensors shapes rectangular (FSR 406) and circular (FSR 402) and how to positioning them inside the coaster figure 6.

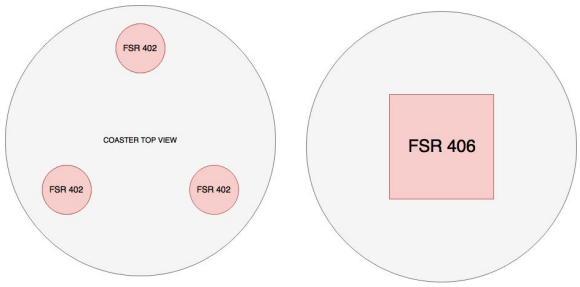

Figure 6: *FSR configurations*

We choose the second configuration because requires only one sensor so it's easier and cheaper.
Then the problem was how to guarantee that the weight of the glass was uniformly distributed over the sensor, so we designed different plates with different tip's dimensions. We found the best behaviour when the tip covers all the sensor's surface.
In order to read how the sensor varies its resistance we used a voltage divider in which the sensor acts as variable pull-up resistor and a 492Ω resistor as pull-down. The measures were taken with the 12 bit ADC of the microcontroller.
Here some graphs, figure 7-8, which show the measurements of and empty and full glass with respect to the tips shape using the same sensor (*FSR 406*).

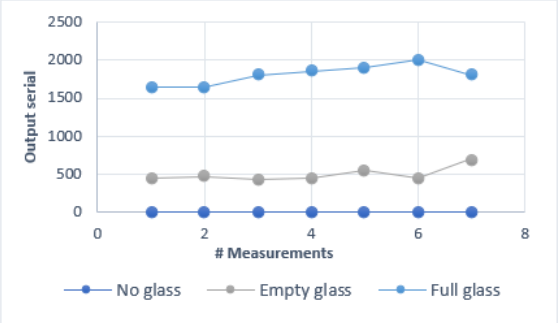

Figure 7: *Measurements with a 37x37 mm tip*

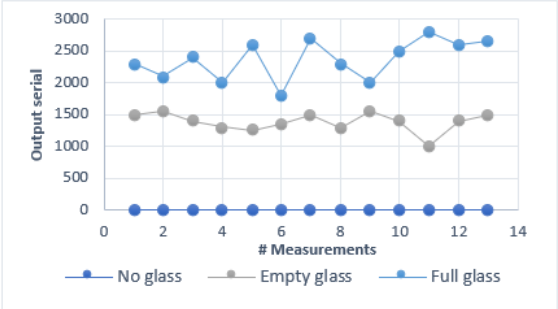

Figure 8: *Measurements with a 18x18 mm tip*

The main difference between the two tip's geometries is given by the fact that more the pressure is localized (figure 8) more its response is unpredictable and then the difference between full and empty glass is not well defined. For this reason and for the reason that a larger support gives greater stability we choose the biggest one (*FSR 406*).

ii. Piezo-electric sensor

To minimize the consumption of the device, it was necessary to introduce a sensor that wakes up the coaster only when requested. A mechanical switch could have been used but it was too difficult for us to include inside the coaster because it needs a leverage systems that is difficult to design in order to guarantee a predictable response. While, a piezo-electric sensor, does not need a leverage system but

only a tip that transfer the impulse force generated when the beer glass hit the coaster.
The initial piezo tested was a *Meas* vibration sensor, figure 9. It works well when it is flexed but it needed a quite long travel (almost 5 mm) in order to give us a satisfactory output. This couldn't be applied to our project without increasing the height of the coaster.

Figure 9: *Meas vibration sensor*

A better result for our purpose was achieved using a piezo-electric disc, usually used in some musical instruments. After some tests we saw that it better captured the moment when the glass hit the coaster. But we noticed that the response was quite unpredictable during our tests we measured that the peak of the signal changed between -10 V and $+10$ V. These peaks can damage the electronics when they are major that the supply voltage moreover when the peaks does not reach the logic threshold voltage they are not detectable from the micro-controller so, a conditioning circuit is needed. The following figure 10 shows the best piezo response that we obtained using a prototype coaster and a half liter beer glass full of water.

iii. Conditioning circuit

The conditioning circuit that we made is composed by one rail-to-rail JFET amplifier configured as in figure 11. This one amplify the signal to the supply voltage (+3.3V) when the

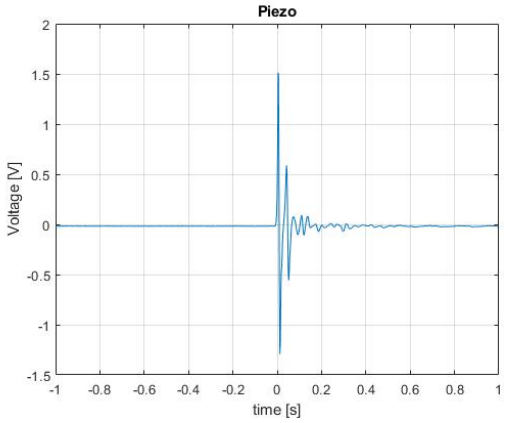

Figure 10: *Piezo-electric disc best response unconditioned*

input signal is major than a offset voltage of 35 mV and also when the input signal is major than the supply voltage. The drawback is the constant consumption of the JFET amplifier that in our case is 750μA.
In order to correctly design the conditioning circuit we used *LTSpice* as simulation tools.
The piezo-electric sensor produces a current that is very low so, the feedback resistor and capacitor have to be design correctly otherwise the input signal will be dissipated through the feedback. So we modeled this physical phenomena as a voltage generator, which simulate the piezo's signal, and a current limiter diode which simulate the finite charge that the sensor generates.
Here some graphs that show the model that we used (figure 11) and the corresponding conditioned output (figure 12).

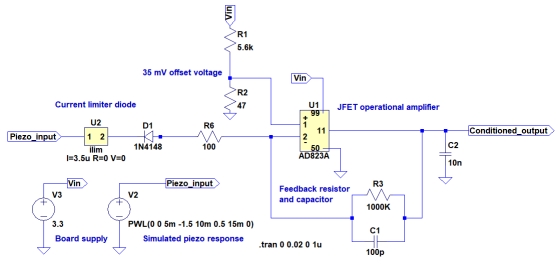

Figure 11: *LTspice schematic*

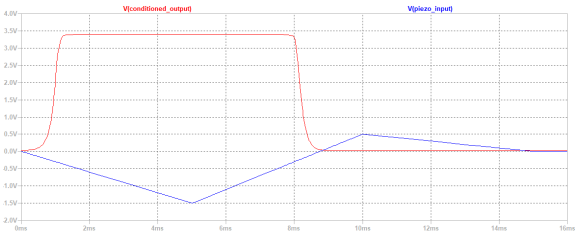

Figure 12: *LTspice simulation*

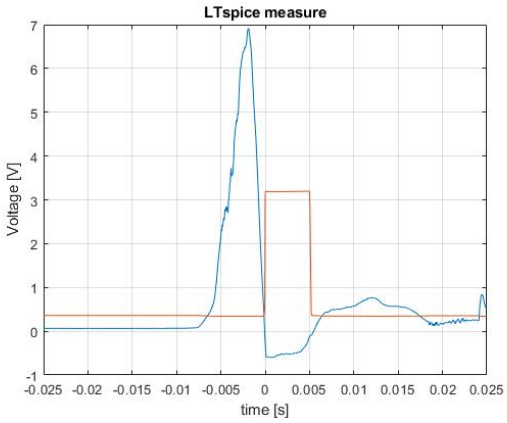

Figure 13: *Real behaviour*

The blue line represent the simulated voltage spike coming from the piezo-electric sensor. This one represents the average behaviour of the measures that we acquired with the oscilloscope. The green line represents the conditioned output. As we can see before the conditioning the voltage spike (blue line) does not reach the logic threshold so, this cannot be recognize as interrupt by the GPIO, while, after the conditioning, the signal (green line) clearly reaches the HIGH logic value. Again, the conditioned signal now clearly varies its state from LOW to HIGH and then from HIGH to LOW. After some circuit's tests we found that the simulation fits quite well the real behaviour figure 13, as shown only the negative peak passes through the conditioning, again the negative peak is not high as the positive one, as in figure 10, because all the current that the sensor produces is smooth down through the feedback capacitor , figure 11.

iv. Button

As waiter's button we choose a small *SPDT* (Single Pole Double Throw). It combines small size and light weight with low cost and extended life. Its function is to generate a positive step signal to one of the micro-controller GPIO when it's pressed. This signal is then associated to a subroutine that send the message "*waiter is needed*". In parallel to the the button we added a 100 nF debouncing capacitor.
The reset button allow us to force the reset mode of the micro-controller while we have to program it.
The last one is used to switch on/off our device and it's connected between the positive terminal of the battery and the board power.

v. Peripherals

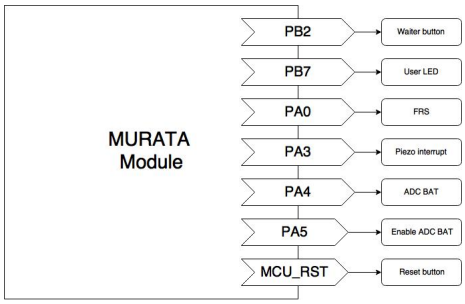

Figure 14: *GPIO ports used*

GPIO The GPIOs or General Purpose Input/Output that we used are listed in the figure 14. As shown some GPIO are configured as ADC (battery level and FSR) some as digital ports (LEDs) and some as interrupt (waiter button, reset button and piezo interrupt). The ADCs and the interrupts work even if the micro-controller is in *sleep mode*.

UART The UART or Universal Asynchronous Receiver-Transmitter peripheral is the one used to debug the software. This function is initialized in the I-CUBE-LRWAN package (section i).

IV. Software

The intelligence of our product is given by a custom software. Before implement it we design its flow chart which is shown in figure 15.

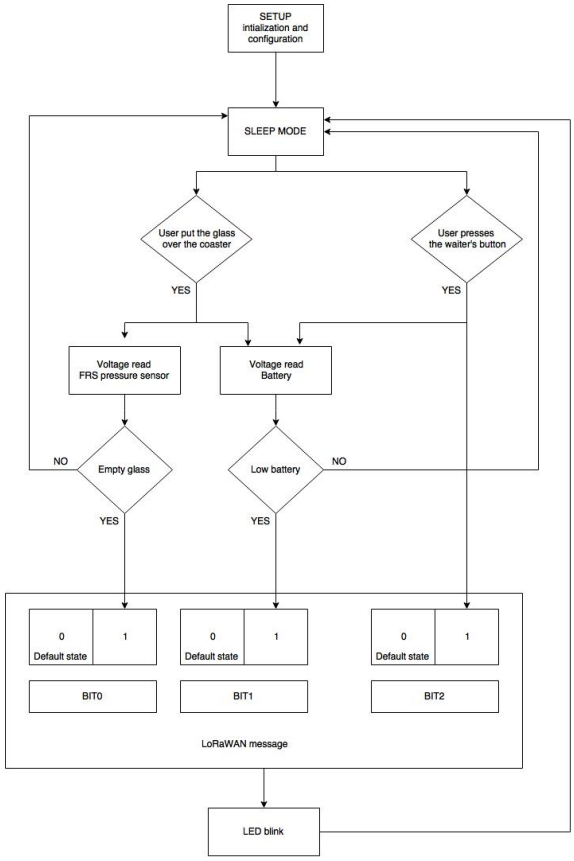

Figure 15: *Software flow chart*

The main code structure is composed by:

- The initialization: Done when when the micro-controller turns on.
- The Sleep mode: The device remain in the sleep mode untill an interrupt occours. This mode reduces the consumptions but at the same time all the peripherals remain active.
- The wake up: The device wakes up from the sleep mode when the beer glass is positioned above it or the user press the *call waiter* button; in the first case the piezo acts as an interrupt which wakes up the micro-controller and, after few microseconds, enable the ADC reading of the FSR. If the ADC value is in the empty glass range the device sends the message *empty glass* (1 empty - 0 full glass: default state) through the LoRaWAN protocol; in the second case instead, the user can call the waiter just by pressing a button and the device sends the message *waiter is needed* (1 waiter is needed - 0 waiter is not needed: default state). For both of the cases when a message is sent also the battery status (1 low battery - 0 full battery: default state) is sent. After that a led blinks and then the device returns to the sleep mode.

V. LoRaWAN

LoRa is a wireless telecommunication network designed to allow long range communications at a very low bit-rate. LoRaWAN defines the communication and security protocol that ensures the interoperability with the LoRa network.

i. I-CUBE-LRWAN package

The use of the *LoRaWAN* protocol was possible through the *I-CUBE* software expansion, which has the following characteristics:

- Application integration ready
- Easy add-on of the low-power LoRa solution
- Extremely low CPU load
- No latency requirements
- Small STM32 memory footprint
- Low-power timing services provided
- Privacy of communication

This extension provides a library of functions already implemented for the micro-processor that we used, allowing it to be exploited in the best possible way. On the other hand this library uses almost all the GPIO.

ii. Network architecture

The network architecture is based on the connection through a gateway to the network

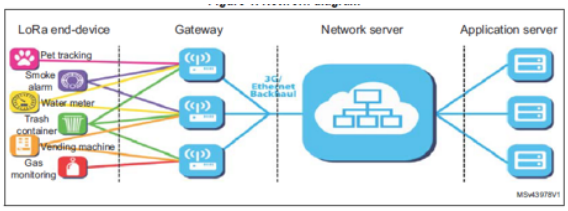

Figure 16: *LoRa network architecture*

server. In our case:

- Lora End Device: One or more smart coasters
- Gateway: Raspberry Pi
- Network server: The Things Network
- Application Server: EndNode

iii. End Device classes

The LoRaWAN protocol has several different classes of end-point devices, addressing the different needs reflected in the wide range of applications. The main differences between the different classes are due to the type of connection that the user want to establish and the consumptions. In figure 18 the main differences between the classes.

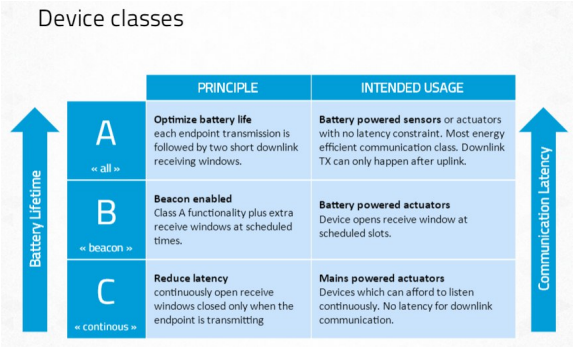

Figure 17: *Differences between the classes - courtesy of LoRa-alliance.org*

Class *A* was considered the most appropriate for our requests for the following reasons.

- Class A operation is the lowest-power end-device system
- Each end-device uplink transmission is followed by two short downlink receive windows
- Downlink communication from the server shortly after the end-device has sent an uplink transmission
- Transmission slot is based on own communication needs of the end-device (ALOHA-type of protocol)

iv. End-device activation (joining)

There is two possible End Device activations:

OTAA (Over-The-Air-Activation): The LoRa end-device and the application server share the same secret key known as AppKey. During a joining procedure, the LoRa end-device and the application server exchange inputs to generate two session keys:

- a network session key (NwkSKey) for MAC commands encryption
- an application session key (AppSKey) for application data encryption

ABP (Activation By Personalization): In the case of ABP, that it was used in this project, the NwkSkey and AppSkey are already stored in the LoRa end-device that sends the data directly to the LoRa network.
We choose ABP because it's easier to connect to the network and it's faster with respect to OTAA. The disadvantage of this method is that it weakens the security. In fact, as we have already said, the encryption keys enabling communication with the network are preconfigured in the device. In the following figure 18 the ABP.

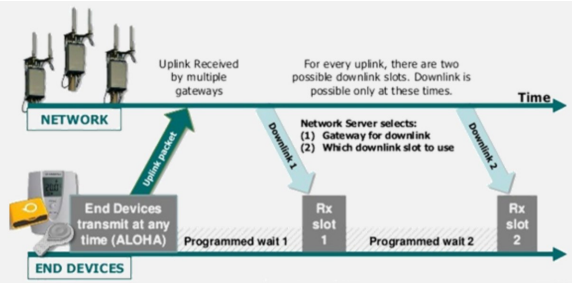

Figure 18: *ABP downlink receiving windows - courtesy of Semtech.org*

VI. Power supply

i. Power consumptions

In this section we explain how we managed the power consumptions.
To find the total power consumptions we used an oscilloscope with a 22Ω shunt resistor connected between the power ground and the board ground. By measuring the voltage between the shunt's terminals we were able to find the current consumed by the micro-controller when it was in the sleep mode, during ADC acquisitions and during LoRa sending.
The first graph in figure 19 shows the current drawn when the micro-controller read the weight over the coaster and then send the message; instead in the second figure 20 is shown the *call waiter* operations. In both of these operations there are the LoRa consumptions. The third figure 21 shows the consumptions of the MCU during the waiter's call. As shown there's one big central step that is due to the LoRa state machine.

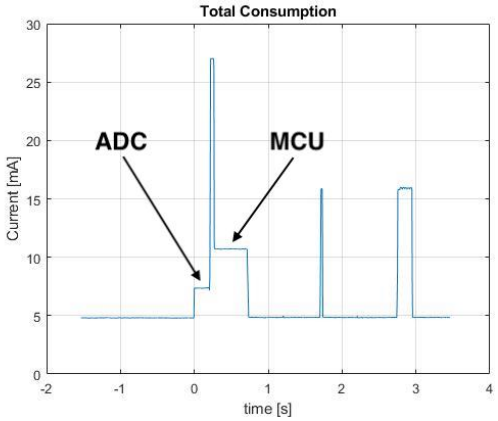

Figure 19: *FSR and LoRa consumptions*

The consumptions of our board only related to LoRa are shown in figure 22.

The first highest peak in figure 22 represents the transmission (Tx) instead the last two the receptions (Rx).
In both of the above plots can be seen that during the sleep mode the consumptions are more or less 5 mA this is mainly due to the JFET amplifier that is very high. So, for this reason we

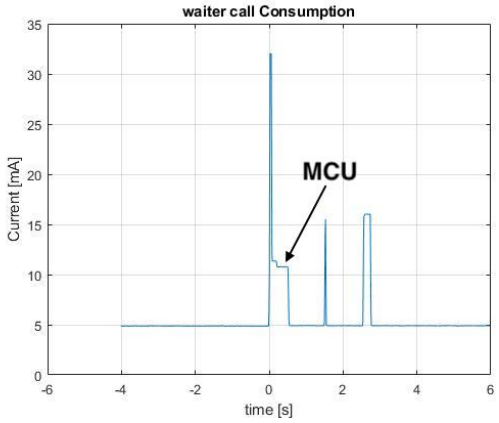

Figure 20: *Waiter call and LoRa consumptions*

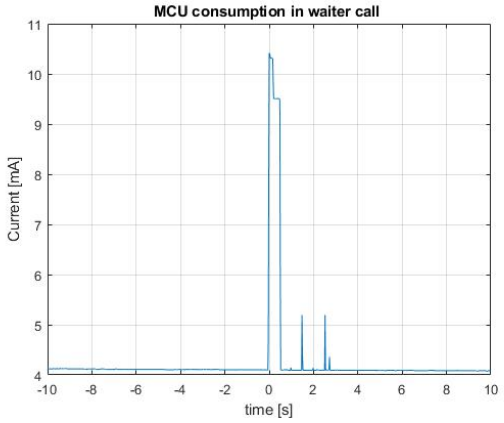

Figure 21: *MCU consumptions during waiter's call*

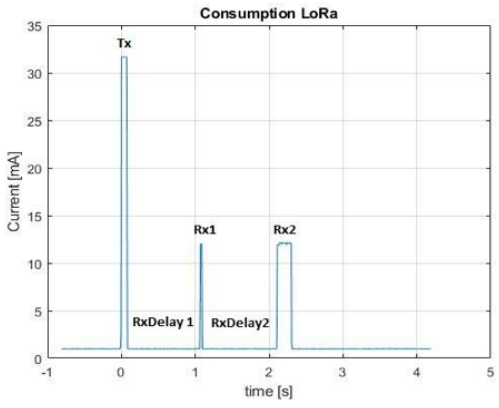

Figure 22: *LoRaWAN consumptions for a class A device*

latert changed the op-amp with another one with a very low power consumption (750μA) but almost same features. The step after the Tx peak, which is present in both graphs but not in figure 22, is related to the micro-controller consumptions.
The main difference between the two plots is related to the fact that if the glass is put on the coaster, figure 19, there is a first consumption's step that is related to the ADC reading while, during the waiter's call, there is not.

ii. Battery and charging system

To choose which battery use in our device we made some calculations considering a typical use of 8 hours in the worst consumption scenario, so when the ADC is reading and LoRa is sending messages, figure 19.
In the table 1 there are some of the consumption's data that we obtained.

Consumptions data			
Parameter	Symbol	Data	Unit
Worst consumptions	C_{worst}	0.0086	[mAh]
Sending time	T_{send}	4	[s]
Opamp consumptions	I_{opamp}	5	[mA]
Assumptions			
Mean working time	t_{hour}	8	[h]
Number of messages sent per day	num_{day}	500	[#]

Table 1: *Consumption's data*

By looking at the table, the minimum capacity consumed per day is:

$$\begin{aligned} C_{tot} &= t_{hour} \cdot I_{opamp} + num_{day} \cdot C_{worst} \\ &= 44.3 \cdot 10^{-3}[\text{A h}] \end{aligned} \tag{1}$$

Where I_{opamp} is the current drawn by the JFET amplifier and C_{worst} is the capacity consumed by the device every time that it send a message. This one corresponds to the integral of the figure 19. But these data were calculated considering 5 mA of JFET consumptions that was not our configuration. The consumption data with the new JFET current of $I_{opamp2} = 750\mu$A.

$$\begin{aligned} C_{tot} &= t_{hour} \cdot I_{opamp2} + num_{day} \cdot C_{worst} \\ &= 10.3 \cdot 10^{-3}[\text{A h}] \end{aligned} \tag{2}$$

In order to guarantee a working life of at least 7 days between one charge to the other one we choose a 100 mAh LiPo battery. The battery that we choose its quite small $15x18x7$ mm, so we can easily put it inside the coaster. This battery is charged through a Crazyfile inductive charging board (figure 24) and a Microchip Li-Po charger controller, all embedded on a custom PCB.
As shown in figure 23 we added a NPN transistor which allow us to enable the voltage divider when the device exits from the sleep mode.

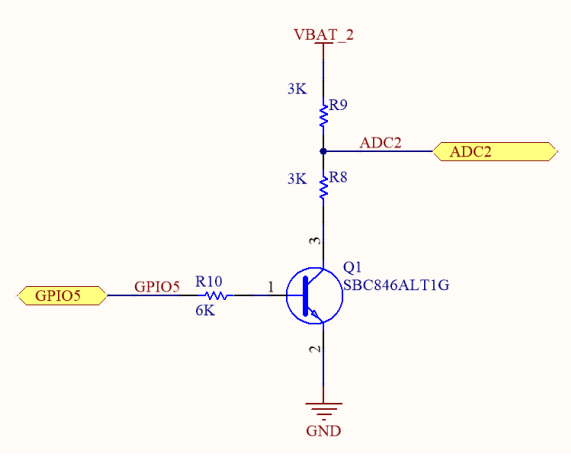

Figure 23: *Schematic of the battery voltage divider and N-transistor*

In order to correctly supply all the electric components we added a LDO (Low-Dropout Regulator) which generates a stable +3.3V supply.

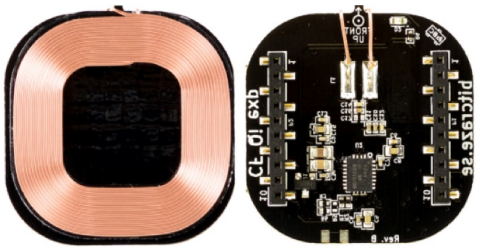

Figure 24: *Crazyflie 2.0 - Qi inductive charging expansion board*

VII. The Things Network

The data relative to the glass beer level, battery charge level and eventually the waiter call are sent from the device to a Rasberry Pi gateway which automatically uploads the package received. Then, thanks to Node-RED it's possible to have access to that package via Raspberry Pi address and then manage all the data received. Node-RED is a free, JavaScript-based, server and web GUI figure 25 used to menage data through a block scheme. The package received contains not only the data related to the coaster but also a unique *ID* so, if more coasters are connected we can easily understand which device is communicating.

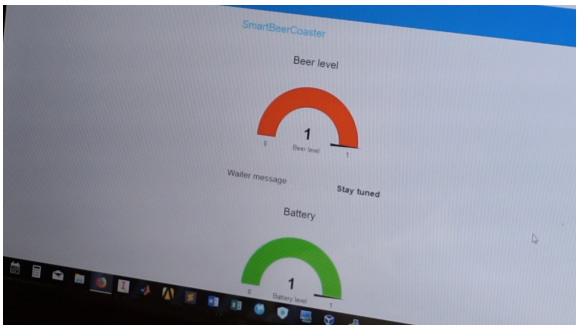

Figure 25: *Web GUI using NodeRed*

VIII. Design

i. 3D printed coaster

Once all the previous parts were chosen we designed the physical structure and we printed it using a *Multimaker 3* and PLA filament.
The final coaster can be seen as a stack where all the layers can be interlocked on each other figure 26.

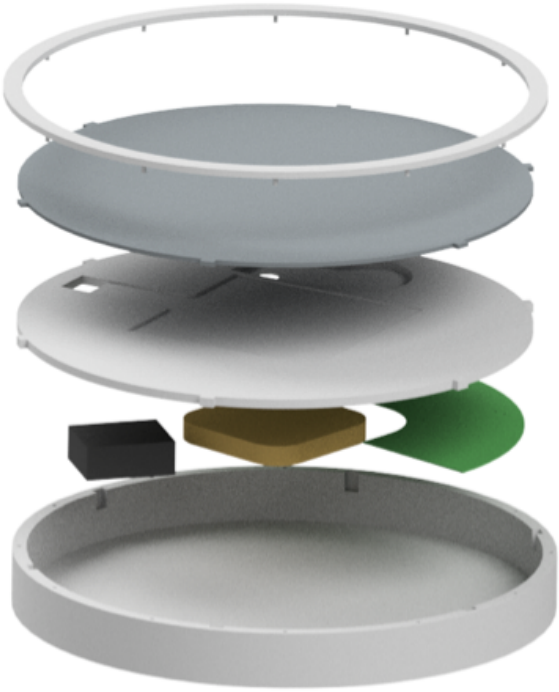

Figure 26: *Coaster's layers stackup*

Starting from the bottom layer up to the top one we have:

- The first layer acts as a support for all the other layers
- In the second layer is positioned all the electronic
- The third layer is used to guarantee a uniform pressure over the sensors
- The last layer allows the fixing of the other ones

Thanks to this modular design is easy to assemble because it doesn't require any tool. The results obtained is shown in figure 26.

ii. Custom PCB

The following figure 27 shows the final PCB layout made with *Altium designer 17*. From the left side there are the power management and the conditioning circuit. On the center there is the micro-controller and on the right side there are the connectors for the buttons and the 868 MHz antenna chip with its matching circuit figure 27. The *PCB* are shaped in order to fit inside the coaster without wasting too much space.

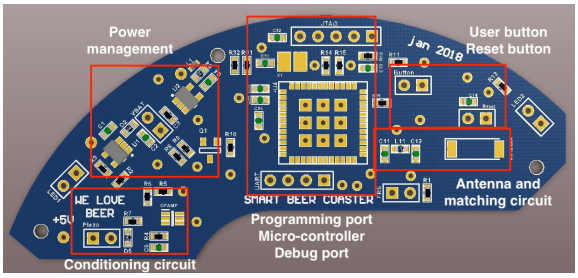

Figure 27: *PCB layout*

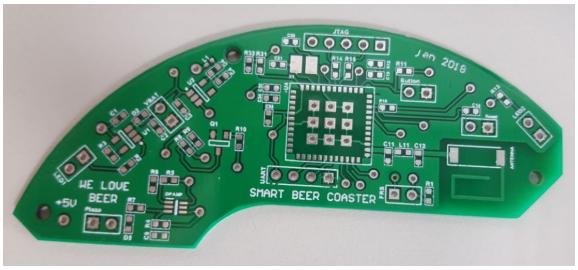

Figure 28: *Final result without components*

IX. Conclusions

The final result of this project is a smart and compact device able to enclose all the features that make it an innovative and useful product.

The main characteristics can be summarized in the following points:

- Ultra low power consumptions
- Micro-controller of last generation
- Built-in LoRaWAN communication protocol
- Built-in battery and wireless charging system.
- Custom PCB design
- Smart design
- Easy to assemble

i. Future improvements

Auto glass calibration: Create a software subroutine that allows the user to calibrate easily the coaster.

Multi glass detections: Create a software subroutine that allows the user to store inside the coaster more than one glass in order to easily use the coaster with different glasses.

GUI application: Dedicated application for an easy and friendly communication.

Multi devices test Test the reliability of the communication of more devices connected to the same gateway.

X. Workgroup

Alessandro Luchetti: firmware development, sensor testing, Internet of Things, CAD design, Minimizing the energy consumption.

Cristiano Strobbe: PCB Design, Power supply, Internet of Things, sensor testing, Minimizing the energy consumption, simulations.

Marco Basilici: firmware development, sensor testing, Internet of Things, CAD Design, Minimizing the energy consumption.

References

[UM2115] User manual, Discovery kit for LoRaWAN: tm: and LPWAN protocols with STM32L0 – www.st.com

[UM2073] STM32 LoRa: registered: software expansion for STM32Cube - www.st.com

[Murata] Hardware design guide- www.murata.com