

GROUP 2

Embedded Systems Design

Magnetic sensor for doors

April 2019

Do Xuan Vy (e1801589)

Tran Minh Hien (e1601116)

Nguyen Quoc Huy (e160)

VAASAN AMMATTIKORKEAKOULU

VAASA UNIVERSITY OF APPLIED SCIENCES

Bachelor of Engineering

# ABSTRACT

Author Tran Minh Hien, Do Xuan Vy, Nguyen Quoc Huy

Title Magnetic field sensor for doors

Year 2019

Language English

Pages 40

Name of Supervisor Jani Ahvonen

This is the report of Final project in Embedded Systems Design class. The goal of this course was to get student familiar with STMicroelectronics line of STM32 microcontroller, as well as developing some small projects by using STM32CubeMX software, Attolic TrueStudio for STM32 and vary of simple components such as LM741 operational amplifier, BC547 transistor, ESP-05 WiFi module …

In more detail for this final project, we will manufacture a magnetic measurement system which can be used to detect whether there is a magnetic field moving near the system. The system is a magnetic field measurement device which can detect electromagnetic fields strength spread from magnet stick on the door and then send wireless messages to subsystem which located one 30 meters away indoor and up to 90 meters outdoor by using XBee. At the first park of the system, Magnetic Field Processing (1), handle analog data from sensor then convert it to Hexadecimal. The second park receive the signal from (1), analysis and display the result on screen and alert user status of the door.

The idea of this product is to establish a security device into the doors, which can confirm if the door is closed and can notify users if they forget to close; and beyond, it can help users control the opening and closing via magnetic lock.

# CONTENT

[ABSTRACT 2](#_Toc7378872)

[CONTENT 3](#_Toc7378873)

[INDEX OF FIGURES 5](#_Toc7378874)

[INDEX OF TABLES 7](#_Toc7378875)

[TERMS AND ABBREVIATIONS 8](#_Toc7378876)

[I. INTRODUCTION 9](#_Toc7378877)

[II. PRE-DESIGN 10](#_Toc7378878)

[1. Block diagram 10](#_Toc7378879)

[2. Components 10](#_Toc7378880)

[a. G-MRCO-028 10](#_Toc7378881)

[b. XBee wireless communication module 12](#_Toc7378882)

[c. STM32 L432KC 13](#_Toc7378883)

[III. HARDWARE 15](#_Toc7378884)

[1. Sensor circuit 15](#_Toc7378885)

[a. Design 15](#_Toc7378886)

[b. Simulation 16](#_Toc7378887)

[c. Connection 17](#_Toc7378888)

[2. XBee connection 18](#_Toc7378889)

[3. Power supply 18](#_Toc7378890)

[4. Testing 18](#_Toc7378891)

[5. Circuit diagram for electronics (PADS Logic) 20](#_Toc7378892)

[a. STM32L432KCU6 development board 20](#_Toc7378893)

[b. G-MRCO-028 sensor 21](#_Toc7378894)

[c. XBee module 21](#_Toc7378895)

[d. LMC6482 OpAmp 22](#_Toc7378896)

[e. Power 22](#_Toc7378897)

[6. PCB design and cooper pour (PADS Layout) 23](#_Toc7378898)

[7. PCB printing and finalize product 24](#_Toc7378899)

[8. Raspberry Pi server 25](#_Toc7378900)

[IV. SOFTWARE 26](#_Toc7378901)

[1. IDE 26](#_Toc7378902)

[2. XBee settings 26](#_Toc7378903)

[3. Client side 26](#_Toc7378904)

[a. Hardware configure with STM32CubeMX 26](#_Toc7378905)

[b. XBee API interface 27](#_Toc7378906)

[c. Minimize power usage 31](#_Toc7378907)

[d. Main function 34](#_Toc7378908)

[4. Server side 35](#_Toc7378909)

[a. XBee receive frame for 16-bit address 35](#_Toc7378910)

[b. Config.py 35](#_Toc7378911)

[c. Server.py 37](#_Toc7378912)

[V. PRICING 38](#_Toc7378913)

[REFERENCE 39](#_Toc7378914)

# INDEX OF FIGURES

[Figure II.1.1: A block diagram of the device 10](#_Toc7378915)

[Figure II.2.1: G-MRCO-028 10](#_Toc7378916)

[Figure II.2.2: G-MRCO-028 pin order 11](#_Toc7378917)

[Figure II.2.3: G-MRCO-028 pin order 12](#_Toc7378918)

[Figure II.2.4: XBee module 12](#_Toc7378919)

[Figure II.2.5: XBee module pin numbers 13](#_Toc7378920)

[Figure II.2.6: STM32 L432KC layout 13](#_Toc7378921)

[Figure III.1.1: Block diagram for the sensor circuit 15](#_Toc7378922)

[Figure III.1.2: Circuit diagram for the sensor circuit 16](#_Toc7378923)

[Figure III.1.3: LTSpice simulating circuit 17](#_Toc7378924)

[Figure III.1.4: LTSpice simulating result 17](#_Toc7378925)

[Figure III.4.1: The whole circuit put on breadboard 19](#_Toc7378926)

[Figure III.4.1: Testing the circuit with a 9*V* battery 19](#_Toc7378927)

[Figure III.5.1: PADS Logic schematic of the system 20](#_Toc7378928)

[Figure III.5.2: STM32 board on PADS Layout 20](#_Toc7378929)

[Figure III.5.2: G-MRCO-028 sensor on PADS Layout 21](#_Toc7378930)

[Figure III.5.3: XBee module on PADS Layout 21](#_Toc7378931)

[Figure III.5.4: LM6482 OpAmp on PADS Layout 22](#_Toc7378932)

[Figure III.5.5: Power supply on PADS Layout 22](#_Toc7378933)

[Figure III.6.1: Bot side PCB with copper pour 23](#_Toc7378934)

[Figure III.6.2: Top side PCB with copper pour 23](#_Toc7378935)

[Figure III.6.3: PCB without copper pour 24](#_Toc7378936)

[Figure III.7.1: Technical sketch of the Starelec box base 24](#_Toc7378937)

[Figure III.7.2: Finalize the product 25](#_Toc7378938)

[Figure IV.3.1: GPIO setting on STM32CubeMX 27](#_Toc7378939)

[Figure IV.3.2: Layout of one API frame 27](#_Toc7378940)

[Figure IV.3.3: Layout of one transmit request for 16-bit address frame 28](#_Toc7378941)

[Figure IV.3.4: XBee.h source code 29](#_Toc7378942)

[Figure IV.3.5: XBee.c source code 31](#_Toc7378943)

[Figure IV.3.6: source code for function disable\_gpio in main.c 31](#_Toc7378944)

[Figure IV.3.7 source code for function enter\_sleep\_mode in main.c 32](#_Toc7378945)

[Figure IV.3.8 key source code in main.c 34](#_Toc7378946)

[Figure IV.4.1: Layout of one receive frame for 16-bit address frame 35](#_Toc7378947)

[Figure IV.4.2: source code in config.py 36](#_Toc7378948)

[Figure IV.4.3: source code in server.py 37](#_Toc7378949)

# INDEX OF TABLES

[Table II.2.1: G-MRCO-028 pin order 10](#_Toc7378950)

[Table III.2.1: Connection from XBee module to STM32 controller 18](#_Toc7378951)

[Table IV.2.1: settings for XBee modules 26](#_Toc7378952)

[Table IV.3.1: Summary of GPIO usage on STM32 controller 27](#_Toc7378953)

[Table V.1.1: List of all hard ware devices 38](#_Toc7378954)

# TERMS AND ABBREVIATIONS

UART Universal Asynchronous Receiver-Transmitter

GPIO General-Purpose Input/Output

SRAM Static Random-Access Memory

OpAmp Operational Amplifier

PCB Printed Circuit Board

IDE Integrated Development Environment

HAL library Hardware Abstraction Layer Library

# INTRODUCTION

The main processor is STM32L432KCU6. XBee wireless module was used to transmit and received data over the air. The magnetic sensor is G-MRCO-028. This report included block diagram of the hardware design and circuit diagram for electronic components. The sleep mode of ARM micro controller and XBee module were also operated so that the system would wake-up arm by itself every single time a signal was transmitted to reduce the power consumption of the battery-powered device.

# PRE-DESIGN

## Block diagram

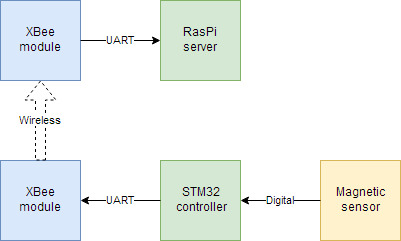


Figure II.1.1: A block diagram of the device

First of all, the team designed a block diagram to preliminary description system. There are 3 main parts: The STM32L432KC, XBee, and magnetic sensor. As can be seen, the diagram above shows the main equipment’s used in the magnetic field processing device, and how they communicate with each other.

## Components

### G-MRCO-028

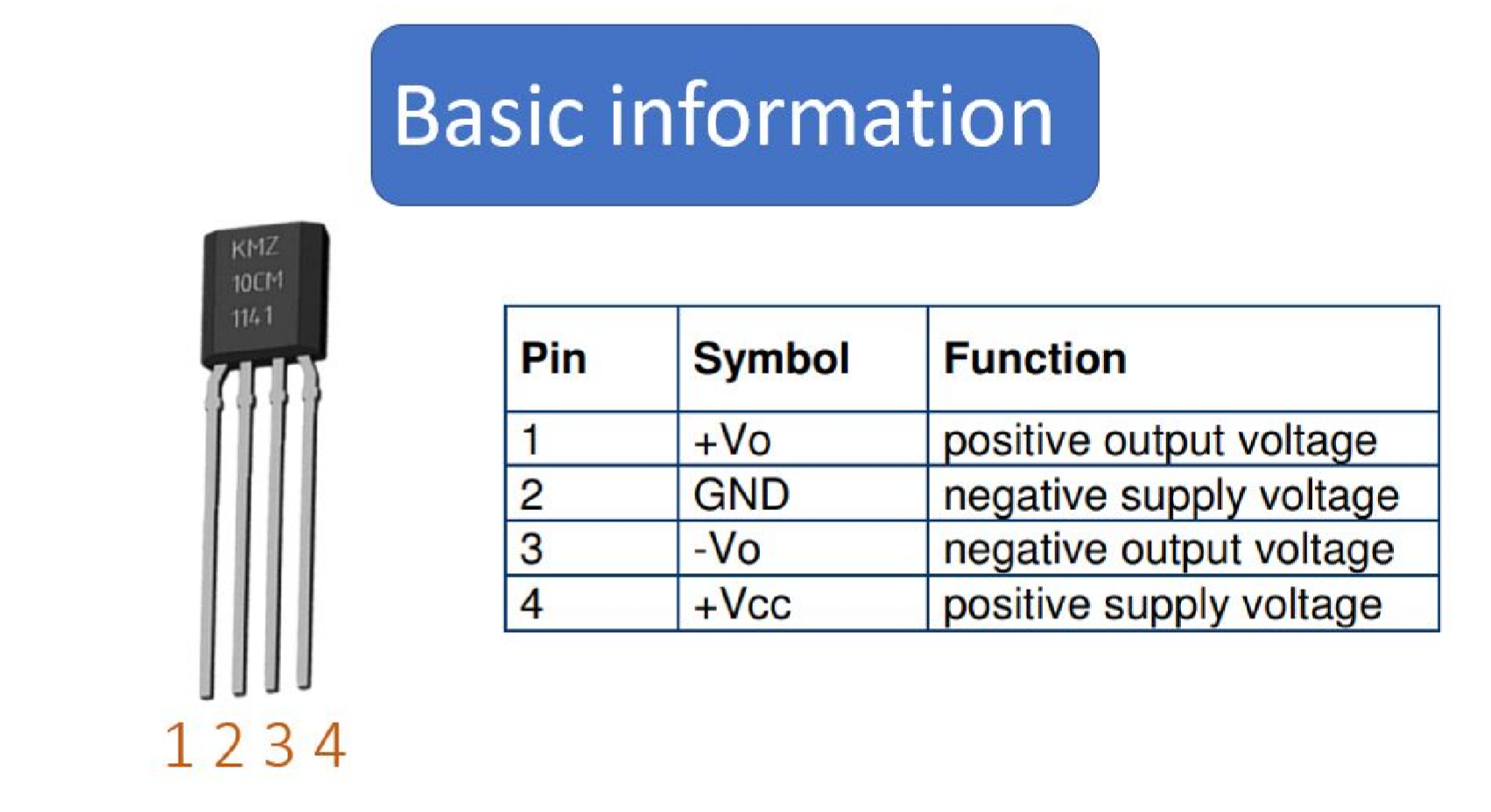


Figure II.2.1:   
G-MRCO-028

|  |  |  |
| --- | --- | --- |
| **Pin** | **Symbol** | **Function** |
| 1 | Vo+ | positive output voltage |
| 2 | GND | negative supply voltage |
| 3 | Vo- | negative output voltage |
| 4 | Vcc | positive supply voltage |

Table II.2.1: G-MRCO-028 pin order

G-MRCO-028 (also known as KMZ10CM) is the name of the magnetic field sensor. This device is use for identifying and measuring magnetic field from surroundings environment. There are numerous different kind of MS in the market but we chose G-MRCO-028 because it has enough features to meet our needs such as:

* Linear signal output
* Over increased field range
* Very low hysteresis
* High sensitivity

Because of its featured properties - high sensitivity and almost no hysteresis – the G-MRCO-028 sensor is used in a wide range of applications, like magnetic field measurement, revolution counters, proximity detection, and position measurement. It is also simple to develop more features for the current device in the future.

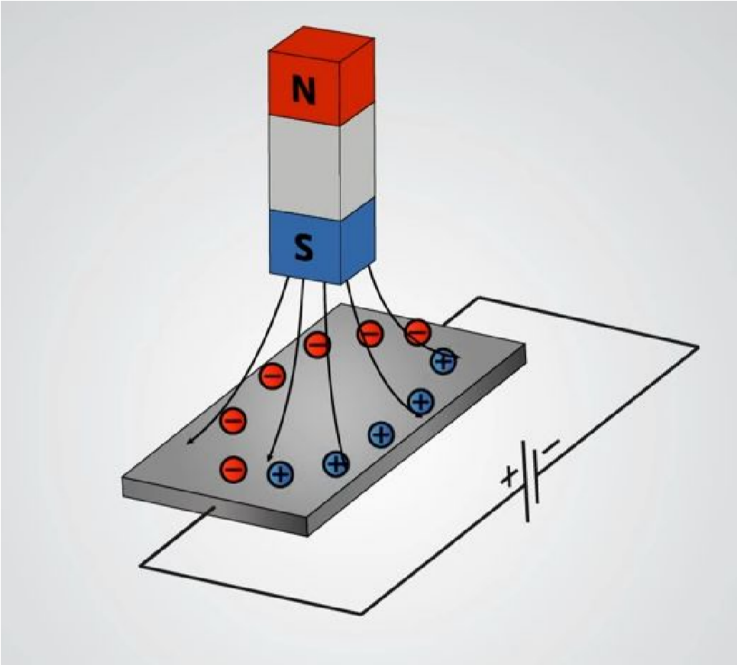


Figure II.2.2: G-MRCO-028 pin order

G-MRCO-028 works based on Hall Effect principle. When a conductor with a running current is placed into a magnetic field, a voltage difference appears perpendicular to the current.

Based on this effect, a Wheatstone bridge is used to detect the magnetic field in the environment. The strength of the magnetic field is determined by measuring the voltage differential between the two bias. In this project, the sensor is connected to the differential ADC on the STM32

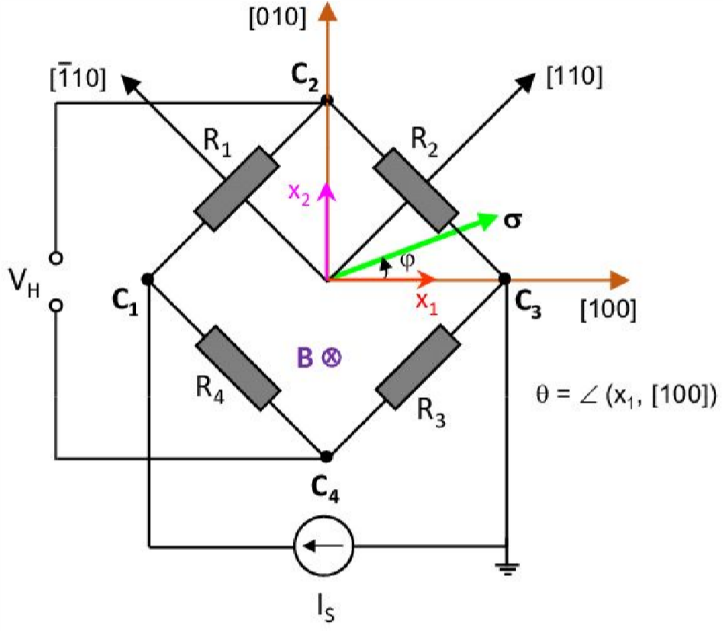


Figure II.2.3: G-MRCO-028 pin order

### XBee wireless communication module

XBee is considered a communication device which transmit data by radio wave in long distance but very low hysteresis. The module was designed to meet IEE802.15.4 standards and support the individual need of low-cost, low-power wireless sensor networks.



Figure II.2.4: XBee module

With range 30m indoor and 90 m outdoor. Also support point-to-point, point-to-multipoint and peer-to-peer topologies supported. XBee is our first choice for dispatching data and communicating from device to device.

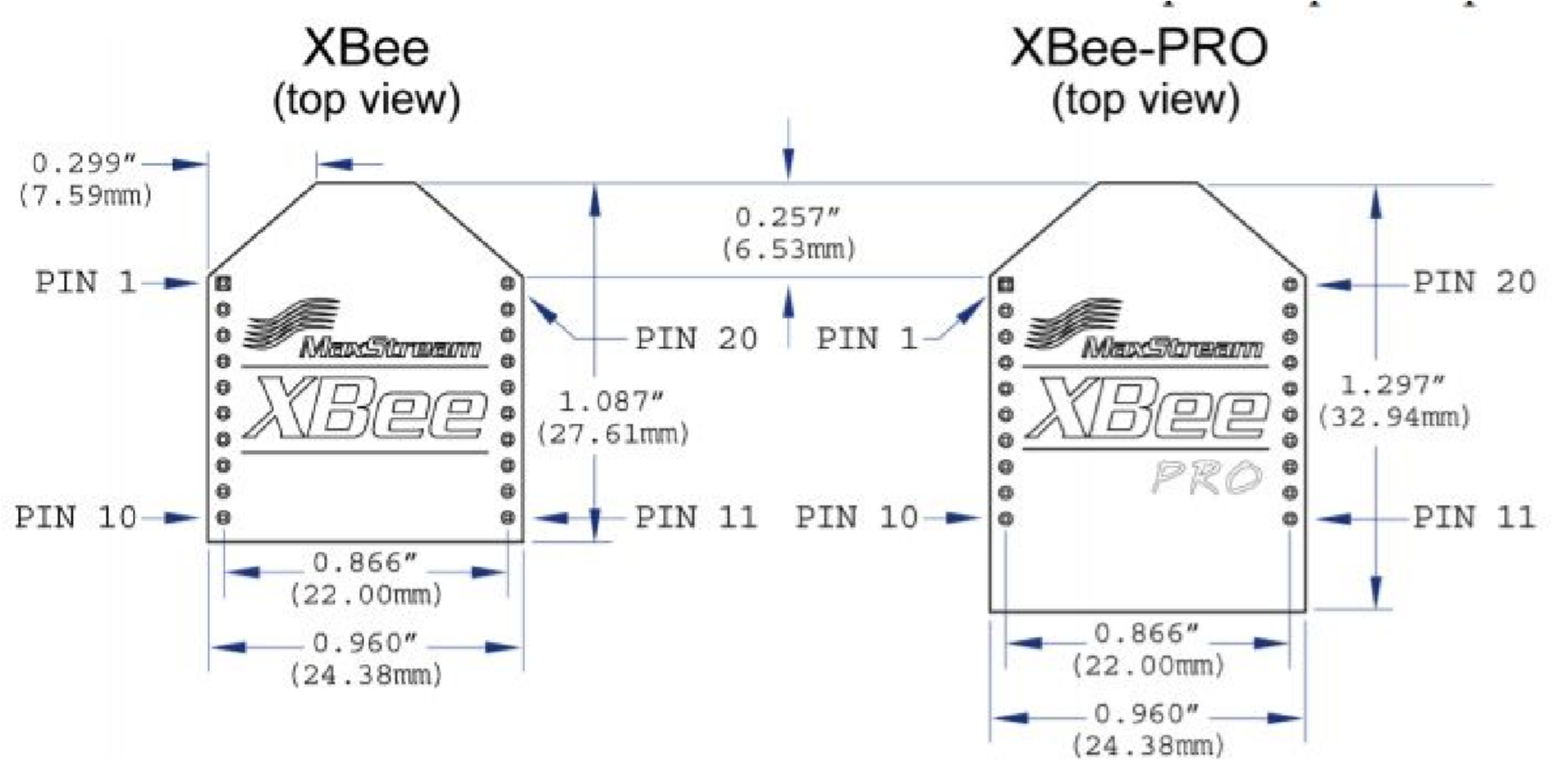


Figure II.2.5: XBee module pin numbers

### STM32 L432KC

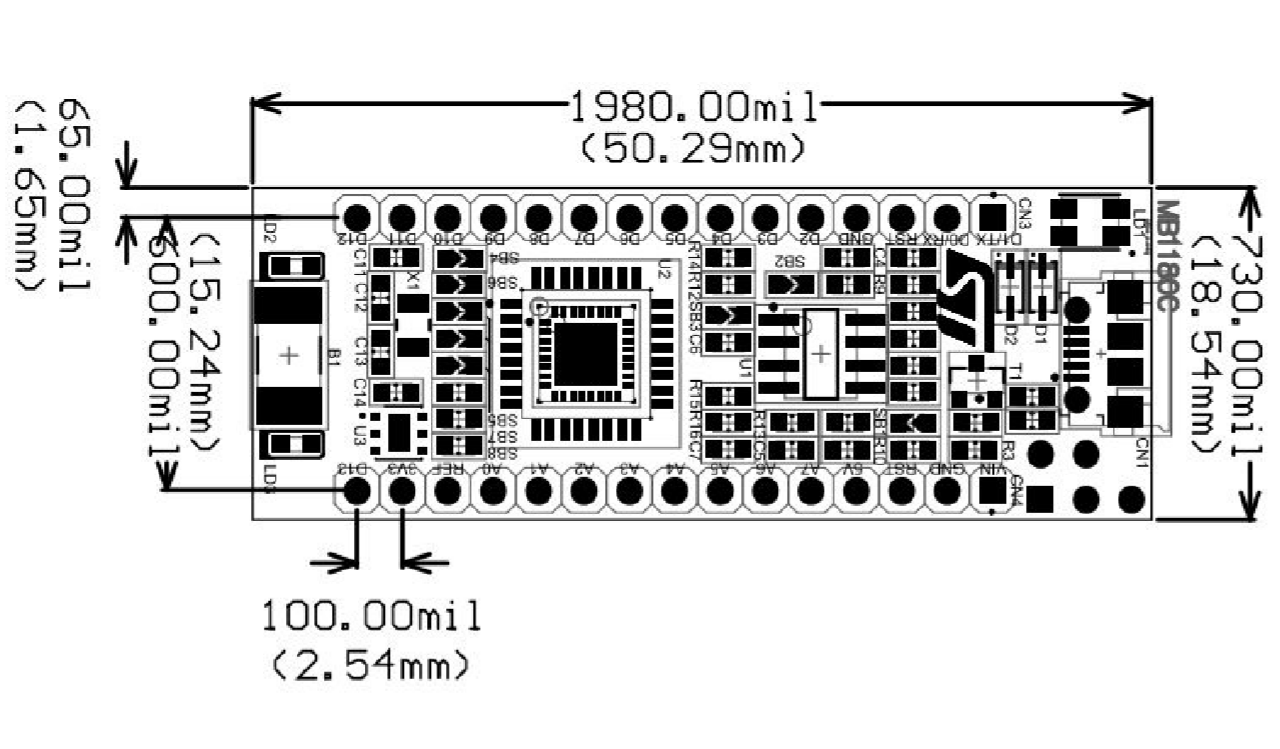


Figure II.2.6: STM32 L432KC layout

Using provided tools to configure and generate driver code is emerging from numerous microcontrollers as a new trend. And one of the famous tools at the moment is STM32Cube-Mx from ST. It helps developer limit down the possible choices when selecting a microcontroller.

The STM32L432 appliance is the ultra-low-power microcontrollers based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 80 MHz. Beside with various outstanding features such as:

* High-speed memories (Flash memory up to 256 Kbyte, 64 Kbyte of SRAM)
* Fast 12-bit ADC (5 Msps)
* Two comparators, one operational amplifier, Two DAC channels
* Low-power RTC
* A general-purpose 32-bit timer

The STM32L432 devices embed numerous protection mechanisms for embedded Flash memory and SRAM: readout protection, write protection, proprietary code readout protection and Firewall, improve the security of the system.

# HARDWARE

## Sensor circuit

### Design

The goal of the sensor circuit is to turn the analog signal from the magnetic sensor into a digital signal that can trigger the STM32 controller to send the data to the server, so that the STM32 can avoid having to pool for the sensor value continuously while still maintain the instancy in notifying the server when the state of the door is changed.



Figure III.1.1: Block diagram for the sensor circuit

This goal is achieve by using two OpAmp circuit: a differential amplifier and a comparator circuit. The differential amplifier will get the differential voltage between the two outputs of the sensor and magnify it. The comparator will compare the value from the previous circuit with a reference voltage value. The reference voltage will come from a voltage trimming circuit.

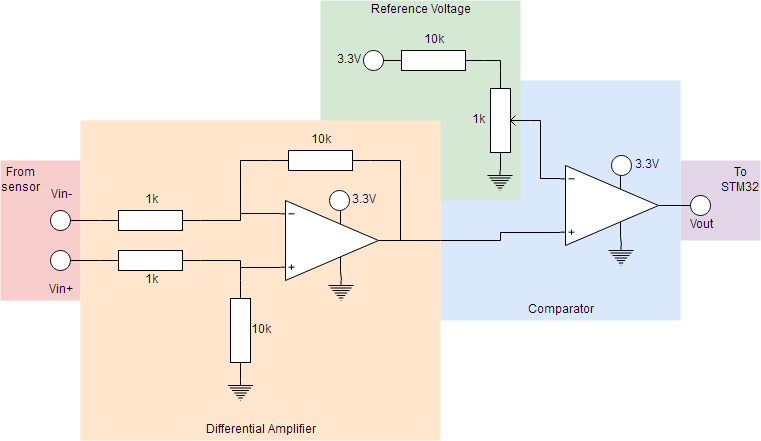


Figure III.1.2: Circuit diagram for the sensor circuit

The output of the differential amplifier will be:

The maximum voltage from the reference voltage will be:

According to experimenting result with available magnets in the lab, the sensor’s output range may go up to about 20 *mV*, which bring the maximum to about 0.2 *V*.

### Simulation

The design circuit was simulated on LTSpice. In the simulation, is replaced by Vsweep, and was replaced by Vbase producing 1.6 *V*. The circuit was then run with DC sweep on Vsweep from 1.6 to 1.7 *V*.

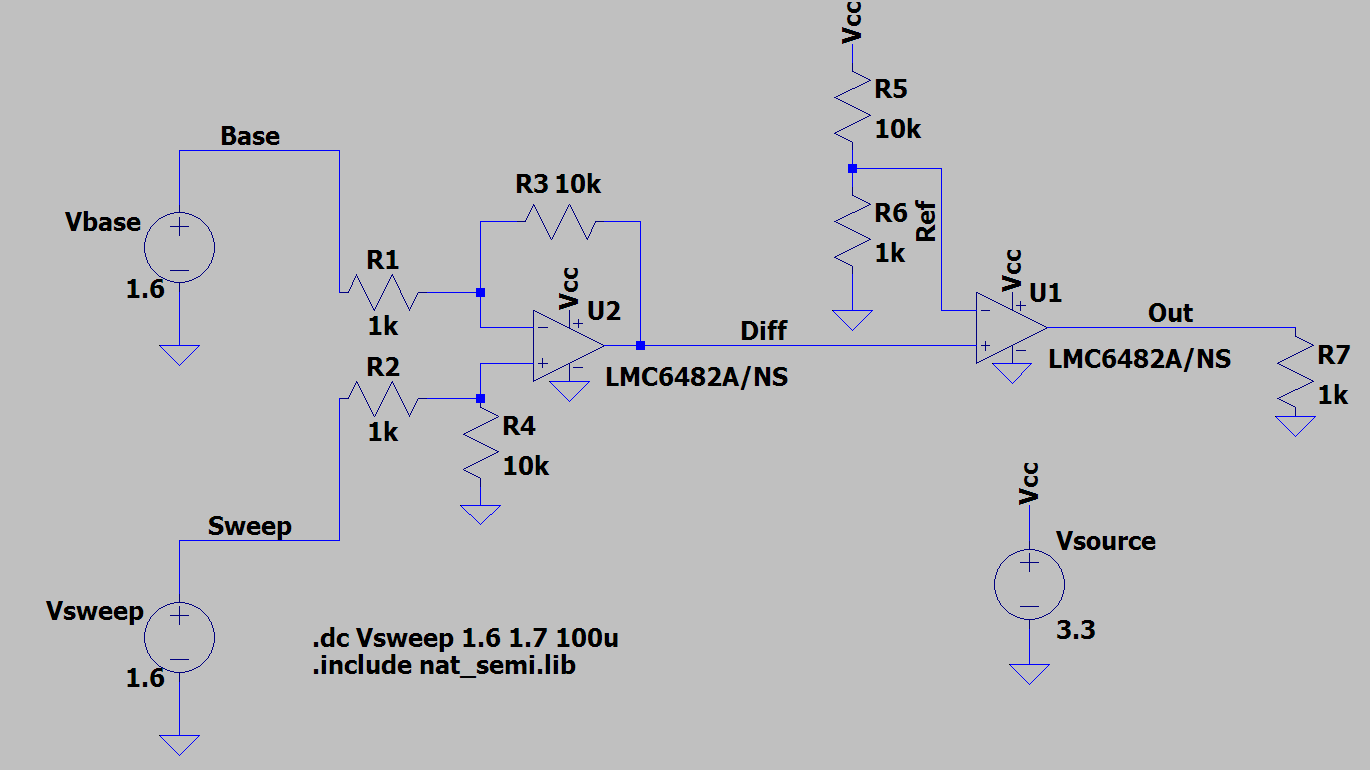


Figure III.1.3: LTSpice simulating circuit

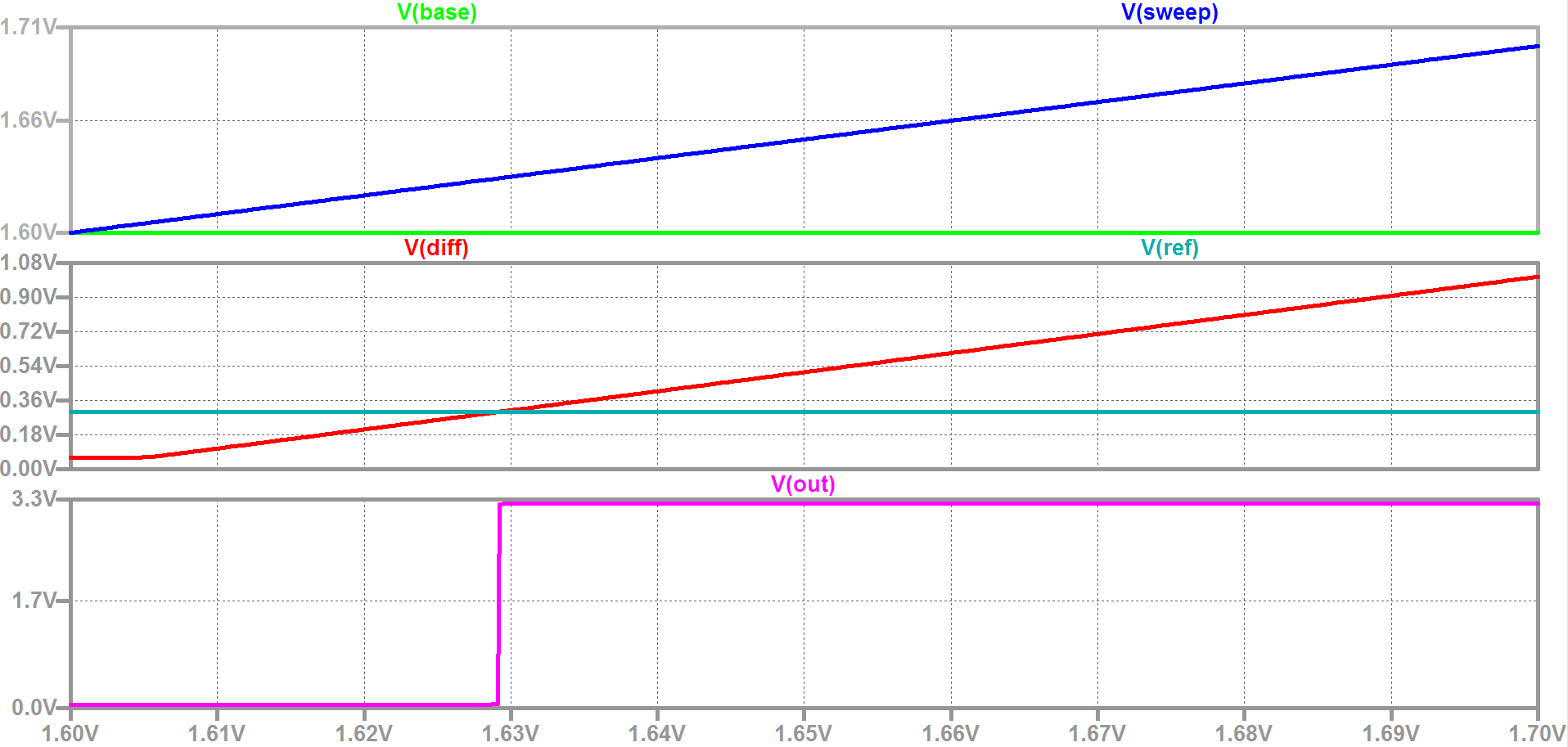


Figure III.1.4: LTSpice simulating result

The result show that the circuit works as designed, except for the fact that there is a idle voltage of about 60 *mV* coming out of Vdiff (output of the differential amplifier). This constitutes the lowest value to be set for.

### Connection

The output from this circuit is connected to pin A7 (PA2)

## XBee connection

The XBee module is directly connected to the STM32. There are five connection needed to be made: Vcc (3.3 *V*), ground, UART transmit/receive, and pin doze (to put the XBee into sleep mode). The connection is specify in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| XBee pin name | XBee pin number | STM32 pin | STM32 GPIO port | Function |
| VCC | 1 | 3.3V |  | Power supply |
| GND | 10 | GND |  | Ground |
| DOUT | 2 | D2/RX | PA10 | UART |
| DIN | 3 | D1/TX | PA9 | UART |
| SLEEP\_RQ | 9 | D9 | PA8 | Sleep mode |

Table III.2.1: Connection from XBee module to STM32 controller

## Power supply

All components on the board will be powered via STM32 controller’s 3.3 *V* power pin, which in turn are stabilize by the STM32’s inner power regulator.

To power the controller, there are three voltage option: 3.3*V*, 5*V*, or 7*V*-12*V*. In this project. At the start, the project was planned to be powered by a series of 3.3*V* button batteries. However, the batteries could not provide the needed current.

Due to this, the 7*V*-12*V* option is used, and the circuit is power with a more powerful 9*V* battery via Vin pin. For the voltage range from 7*V* to 9*V* that the battery can provide, the maximum current draw is 450 *mA*. From experimenting, it was determined that the maximum current consume by the entire circuit does not exceed 130 *mA*.

The connection between the battery and the STM32 is also protected by two decoupling capacitors: one 10 *μF* electrolyte and one 100 *nF* ceramic capacitor. A MF-MSMF050-2 resettable fuse is also used as a circuit breaker.

## Testing

After designing, the whole circuit was put on breadboard and tested.

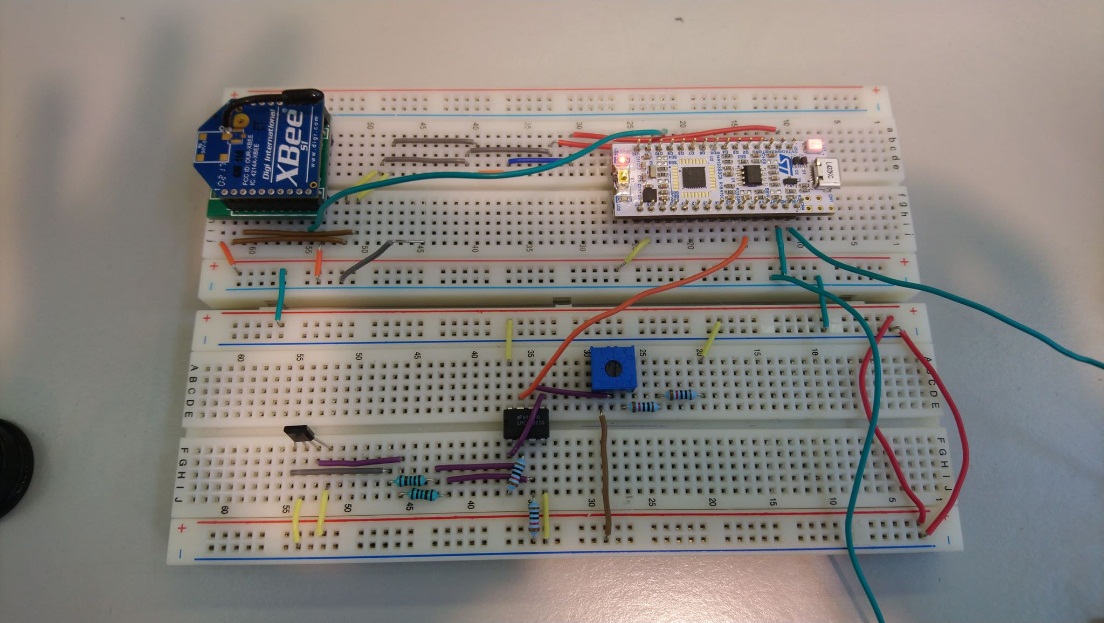


Figure III.4.1: The whole circuit put on breadboard

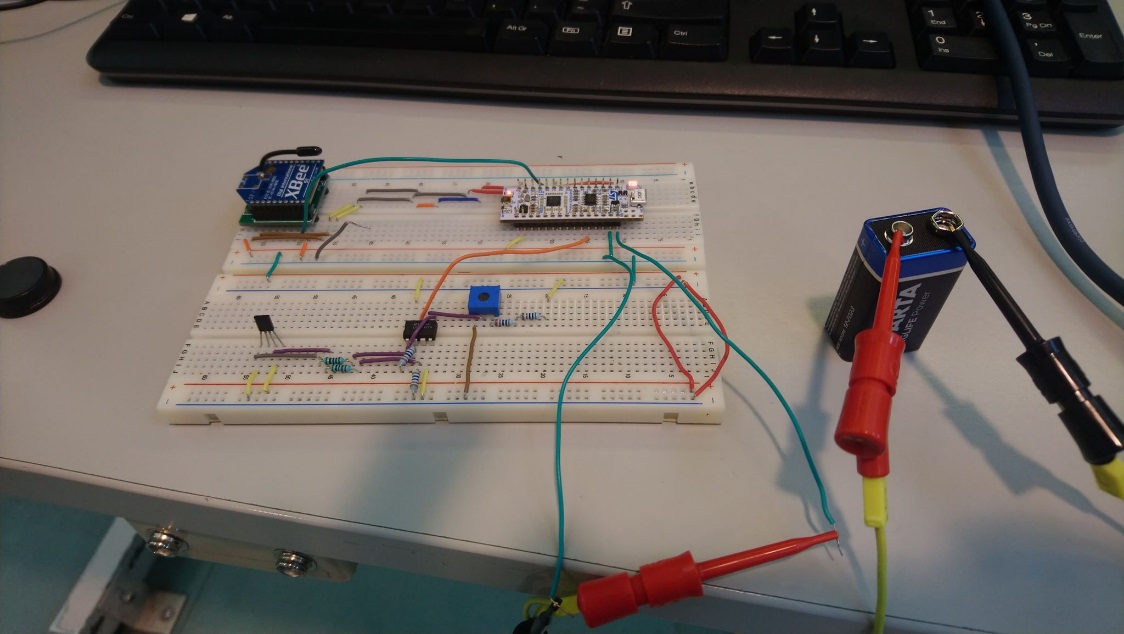


Figure III.4.1: Testing the circuit with a 9*V* battery

The circuit fulfill the following criteria:

* The sensor can detect a magnet and generate the correct signal
* The STM32 can connect to the XBee module
* The STM32 can be powered by the battery with a current stable enough for the UART communication between the STM32 and the XBee

PCB is one of the most important factors, it makes everything simple and neat, for that reason the team has spent a lot of time to design and fix it.

## Circuit diagram for electronics (PADS Logic)

There are 5 parts in this final project system layout:

### STM32L432KCU6 development board

This is the main processor of the whole system, which is small, convenient and powerful for input data processing, data transfer for necessary modules as well as controlling the entire system

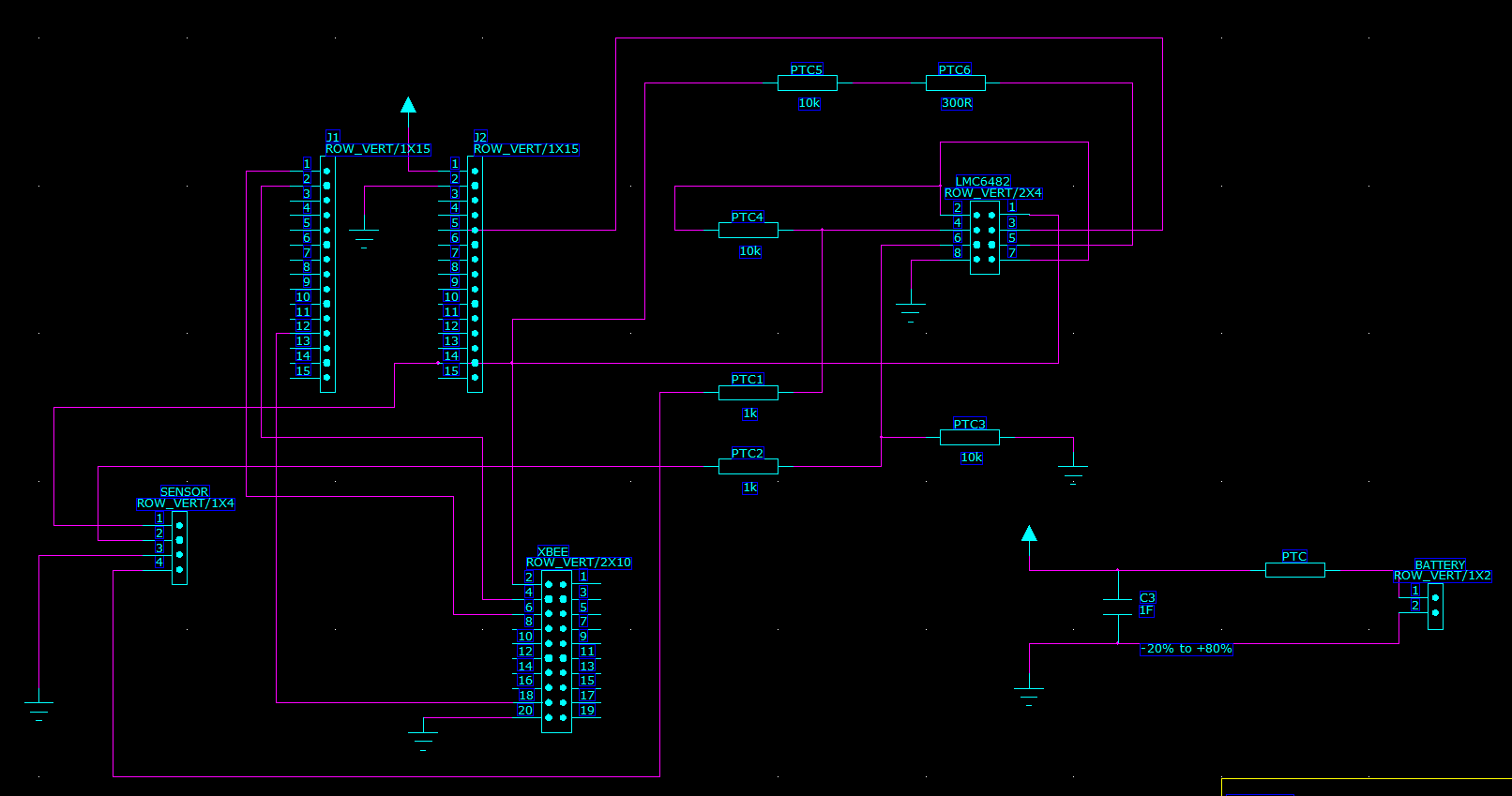


Figure III.5.1: PADS Logic schematic of the system

Because there is not pattern of this kind of components in the library of PADS, we have to design them by ourselves. We used 2 rows 15 pins stand for the STM32 board. The 2 first pins and the 12th pin on the left connected with XBee module. The 2 first pins on the right are connected to 9V power and ground. The 14th pins on the right is the 3.3V power of sensor, XBee and LMC6482.

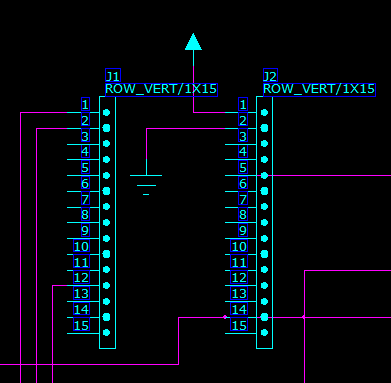


Figure III.5.2: STM32 board on PADS Layout

### G-MRCO-028 sensor

This is the magnetic sensor, which can measure and feedback the signal of change in magnetic field around it.

We used 1 row 4 pins stand for the sensor. The 1st pin is linked with 3.3V power of STM32, the 3rd pin is linked with ground. The 2nd and 4th pins are output voltage which is connected to the OpAmp LMC6482 to gain an amplifier signal output

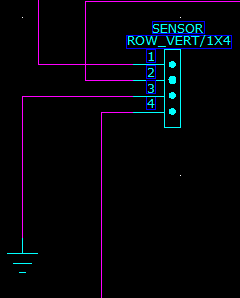


Figure III.5.2: G-MRCO-028 sensor on PADS Layout

### XBee module

This is the wireless module to transmit and received data through radios.

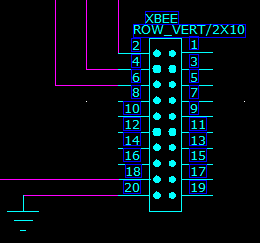


Figure III.5.3: XBee module on PADS Layout

We used 2x10 pins row stand for XBee. The 1st pin on the left is connected with 3.3V power of STM32, the 2nd and 3rd pins on the left are UART data pins, which is linked with STM32. The 9th pin is sleep mode control pin, which is linked with STM32. And the last one connects with ground.

### LMC6482 OpAmp

This is the OpAmp to produce a potential larger than the input terminals. There are 2 OpAmps in this LMC6482

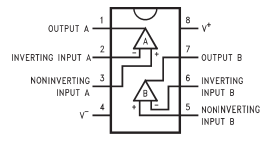
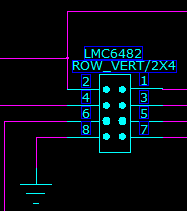


Figure III.5.4: LM6482 OpAmp on PADS Layout

8th pin is connected to the ground. 2nd ,4th ,6th pin is of the first OpAmp. 3rd, 5th, 7th is of the second OpAmp. 1st pin is connected to the 3.3V power of STM32

### Power

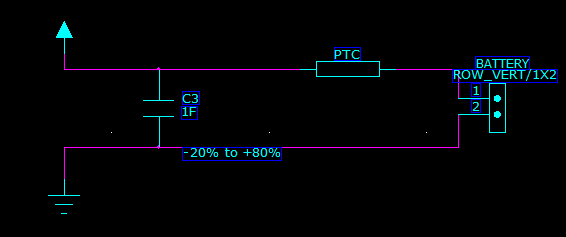


Figure III.5.5: Power supply on PADS Layout

The power system used a 9V battery. We added a PPTC Resettable Fuse at the positive power pin to protect the components of product if there is some power problems. It also got a capacitor between positive power line and ground for capacitive coupling

## PCB design and cooper pour (PADS Layout)

After draw by using PADS layout, we linked the schematic to PADS logic and start to design the PCB.

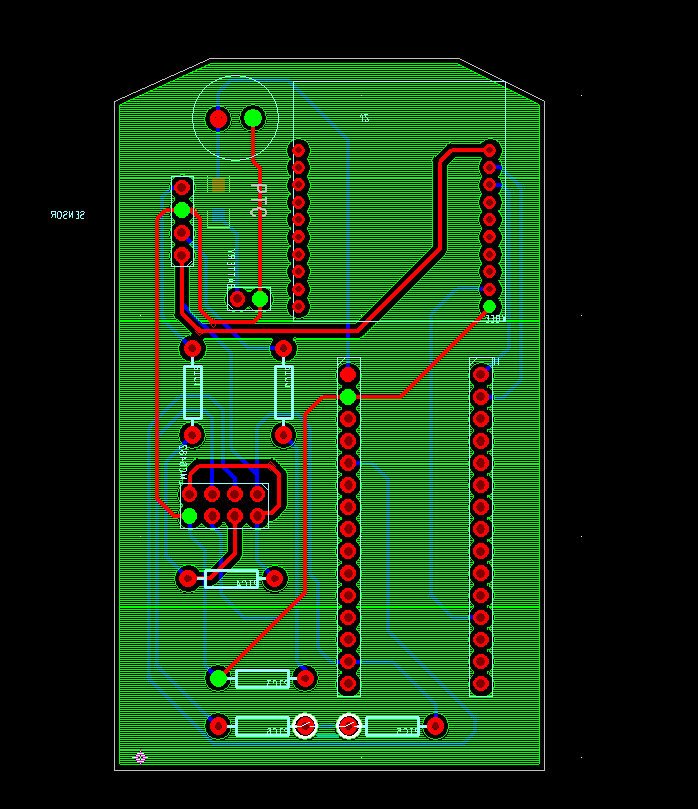


Figure III.6.1: Bot side PCB with copper pour

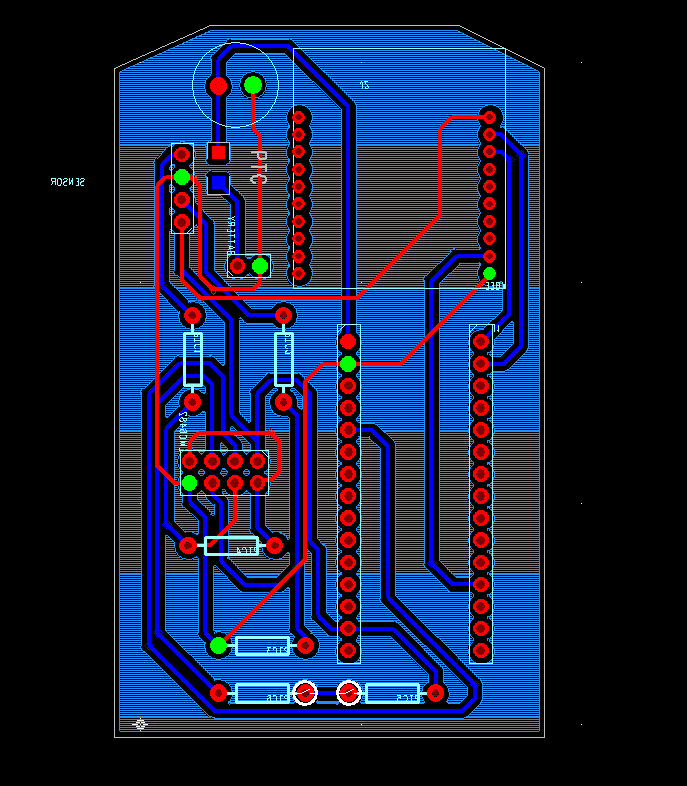


Figure III.6.2: Top side PCB with copper pour

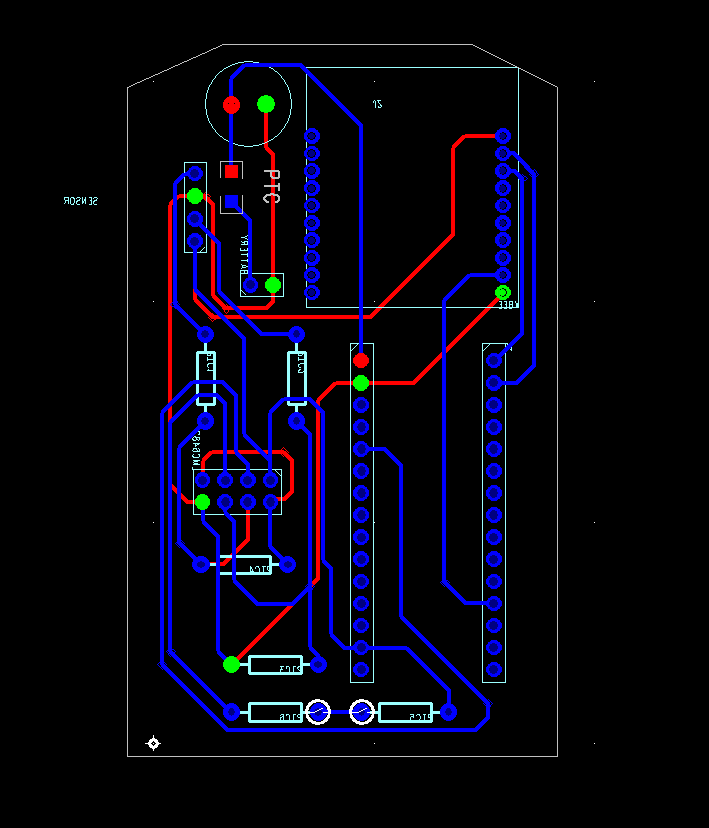


Figure III.6.3: PCB without copper pour

## PCB printing and finalize product

The goal of this project is that the whole system must be fit into the Starelec box 1591A-FL. The team measured the technical data of the equipment and PCB dimensions and tried to optimize the space to fit the project requirements.

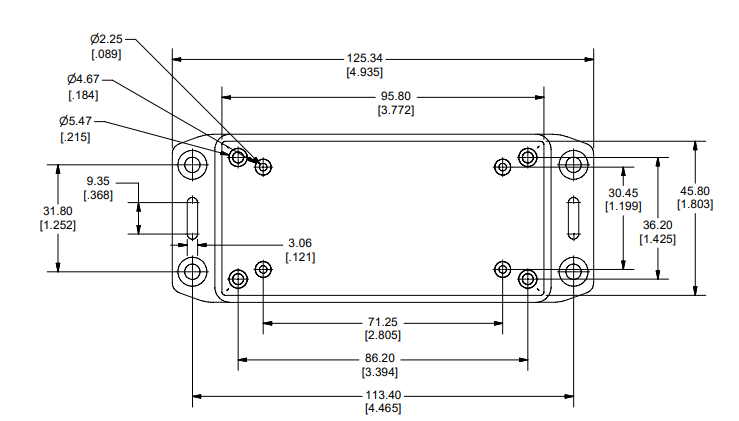


Figure III.7.1: Technical sketch of the Starelec box base

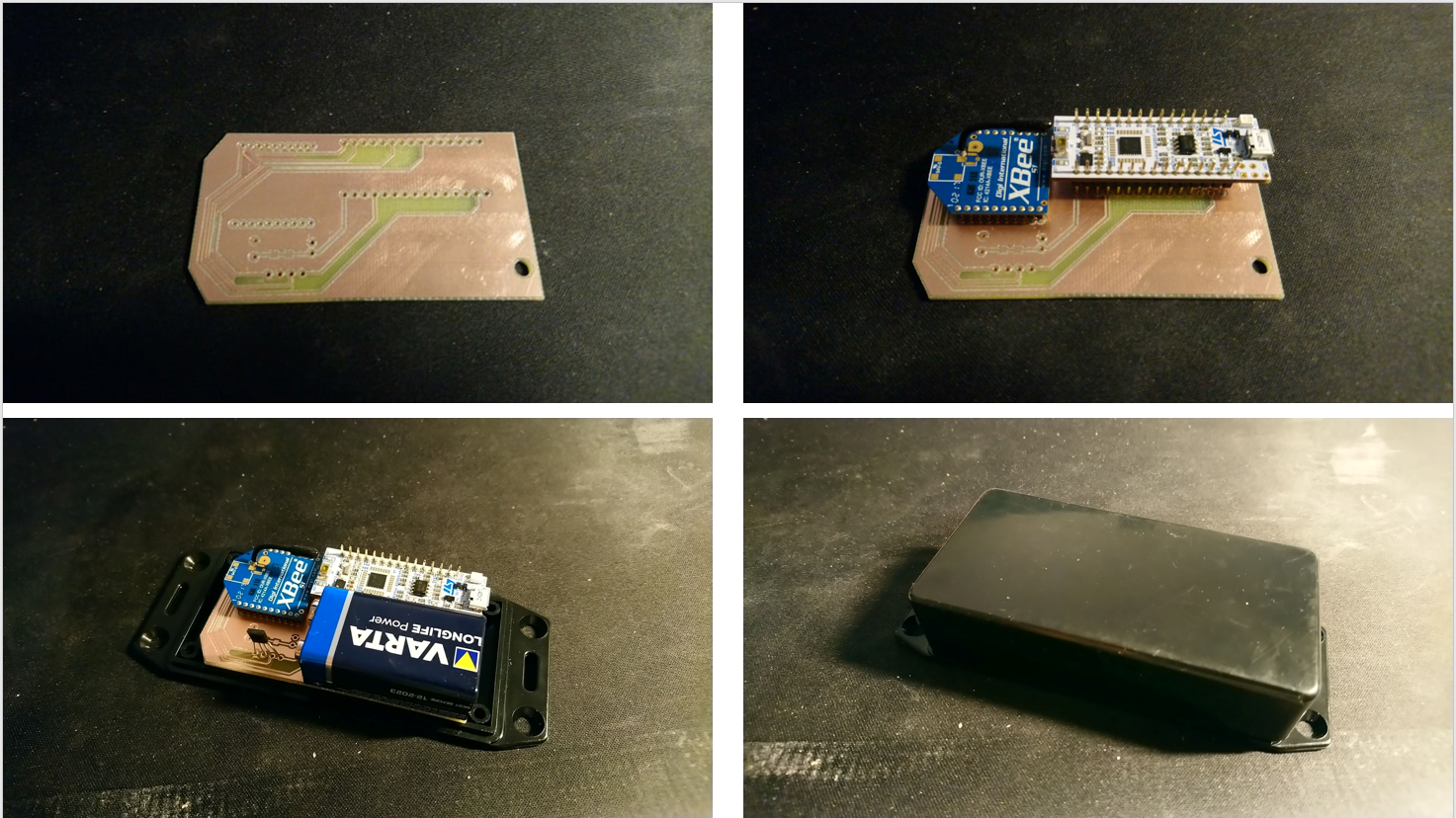


Figure III.7.2: Finalize the product

## Raspberry Pi server

For the server, a Raspberry SoC is used. The XBee module is connected to the Pi’s default UART pins.

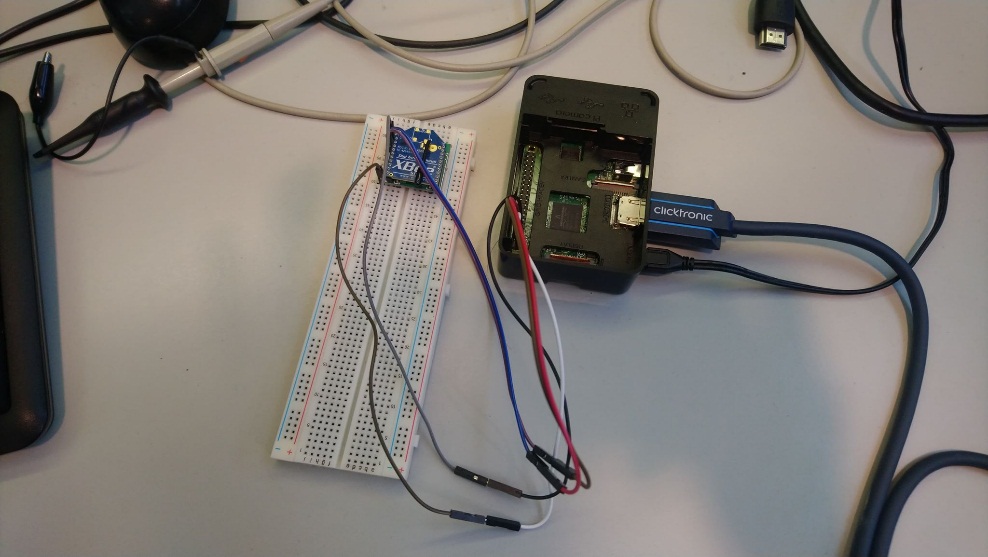


Figure III.8.1: connection between the Pi and XBee module

# SOFTWARE

## IDE

The main software developing tools used in this project are:

* STM32CubeMX: Official code generating application from STMicroelectronics, the main manufacturer of STM32 development boards. Generating code template and configure peripherals on the STM32 board.
* Atollic TrueSTUDIO: an IDE based on Eclipse, GNU gcc and gdb, used to compile the code and flash the STM32 with the compiled .elf file, as well as run and debugging the code on the board via ST-LINK
* Rasbian: The most common OS for Raspberry Pi, providing important application for programming the RasPi server such as nano, python, etc…

## XBee settings

The settings of the XBee is as in the following table

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | AT command | Client XBee | Server XBee |
| 16 bit address | MY | AB02 | AB01 |
| Channel | CN | C | C |
| Sleep mode | SM | 2 (pin doze) | 0 (no sleep) |
| UART baud rate | BD | 3 (9600) | 3 (9600) |

Table IV.2.1: settings for XBee modules

The two modules are set to have two different address. Sleep mode is enable on client XBee module

## Client side

### Hardware configure with STM32CubeMX

All GPIO used in the project is listed in the following table

|  |  |  |  |
| --- | --- | --- | --- |
| Pin name | Pin GPIO port | Pin mode | Function |
| A7 | PA2 | GPIO\_EXTI2 | Receive data from sensor |
| D9 | PA8 | GPIO\_Output | Controll XBee’s pin doze |
| D1/TX | PA9 | USART1\_TX | UART transmit to XBee |
| D2/RX | PA10 | USART1\_RX | UART receive from XBee |

Table IV.3.1: Summary of GPIO usage on STM32 controller

Pin A7 is set in interrupt mode to wake the STM32 when the state of the door is changed. Pin D9 is connected to XBee pin 9 (pin doze) to put the XBee to sleep when there is no communication need. Pin D1/TX and D2/RX are two UART serial pins, connected to DIN and DOUT on the XBee, respectively. The UART interfaced used is UART1, at baud rate 9600, configuration 8-N-1, without flow control.

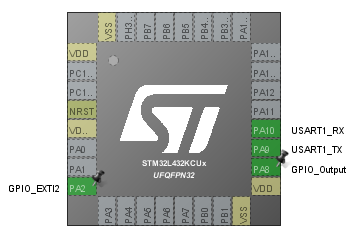


Figure IV.3.1: GPIO setting on STM32CubeMX

### XBee API interface

#### XBee API frame

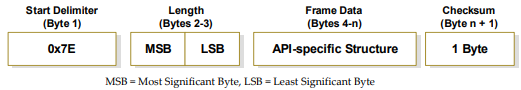


Figure IV.3.2: Layout of one API frame

One frame of XBee API’s communication mode is consisted of four parts:

* Start delimiter (1 byte, 0x7E): signal the start of one frame
* Length (2 byte, big endian): specify the length of the frame data, not including the delimiter, the length itself, and the checksum byte
* Frame data: the data in the frame, has specific structure depending on the API function
* Checksum (1 byte): to verify the integrity of the frame. The checksum is calculated by taking the sum of all bytes in the frame data, take only the lowest 8 bits, and subtract that value by 0xFF. If a frame has invalid checksum, it will be discarded completely by the XBee module

#### XBee transmit request for 16-bit address

This frame will be send from the STM32 controller to the XBee module to initiate sending a packet to the server



Figure IV.3.3: Layout of one transmit request for 16-bit address frame

A transmit request for 16-bit address has five fields in its frame data:

* API identifier (0x01, 1 byte): specify the API function
* Frame ID (1 byte): specify the frame ID for the host to reply with an ACK frame
* Destination address (2 bytes): specify the receiving module’s address
* Option (1 byte): has two flag
  + 0x01: enable the ACK response if set
  + 0x04: send packet with PAN broadcast ID if set
* RF data (maximum 100 bytes): data to be sent

#### XBee.h and XBee.c

Inside the XBee.h and XBee.c files are a few functions taking care of the communication from the STM32 to the XBee client module.

#ifndef XBEE\_H

#define XBEE\_H

#include "stm32l4xx\_hal.h"

#define XBEE\_BUFFER\_LENGTH 100

void init\_XBee(UART\_HandleTypeDef\* huart);

void tx\_req\_XBee(uint8\_t id, uint16\_t addr, uint8\_t opt, char\* data);

#endif

Figure IV.3.4: XBee.h source code

In XBee.h two function are defined to be used in the main program. These two functions are:

* init\_XBee: register the UART handle to be used to communicate with the XBee module
  + Arguments:
    - huart (UART\_HandleTypeDef\*): pointer to the UART handle to be used
* tx\_req\_XBee: make a transmit request to the XBee
  + Arguments:
    - id (uint8\_t): frame ID
    - addr (uint16\_t): destination address
    - opt (uint8\_t): options
    - data (char\*): data to be sent

The definition of these function is shown in the following figure

#include "XBee.h"

#include "stm32l4xx\_hal.h"

#include "string.h"

// global pointer to the UART handle used to communicate with the XBee

UART\_HandleTypeDef\* huart\_g;

// put an unsigned 16 bit number onto a position in an uint8\_t array

void put\_int16(uint8\_t\* pos, uint16\_t num) {

pos[0] = (num & 0xFF00) >> 8;

pos[1] = num & 0xFF;

}

// concatenate a char array into a uint\_8 array at position defined by offset

void concat\_XBee(uint8\_t \*buf, char\* data, size\_t offset) {

for (size\_t i = 0; data[i]; i++) {

buf[i + offset] = data[i];

}

}

// save the huart used to communicate with the XBee

void init\_XBee(UART\_HandleTypeDef\* huart) {

huart\_g = huart;

}

// send one API frame to the XBee, with checksum calculation

// take two arguments: data (uint8\_t array) and len (size\_t)

void send\_XBee(uint8\_t \*data, size\_t len) {

// the buffer of data to be sent

uint8\_t buf[XBEE\_BUFFER\_LENGTH];

// the sum of the frame data values

uint16\_t sum = 0;

// starting delimiter

buf[0] = 0x7E;

// length

put\_int16(buf + 1, len);

// copying the data into the buffer and taking the total sum of the frame data values

for (size\_t i = 0; i < len; i++) {

buf[i + 3] = data[i];

sum += data[i];

sum &= 0xFF;

}

// calculate the checksum

buf[len + 3] = 0xFF - sum;

// send the buffer to the XBee via UART

HAL\_UART\_Transmit(huart\_g, buf, len + 4, 100);

}

// send one API transmit request to the XBee

void tx\_req\_XBee(uint8\_t id, uint16\_t addr, uint8\_t opt, char\* data) {

// the buffer contain the frame data

uint8\_t buf[XBEE\_BUFFER\_LENGTH];

// API identifier

buf[0] = 0x01;

// frame ID

buf[1] = id;

// destination address

put\_int16(buf + 2, addr);

// options

buf[4] = opt;

// copying the data to be sent into the buffer

concat\_XBee(buf, data, 5);

// send the frame data to the XBee

send\_XBee(buf, strlen(data) + 5);

}

Figure IV.3.5: XBee.c source code

### Minimize power usage

To minimize power usage, the following methods was taken

#### Put unused pins into analog mode

Unused pins can be set in analog mode in the STM32CubeMX

#### Disable unused GPIO

void disable\_gpio() {

// GPIO struct to be set to GPIO ports

GPIO\_InitTypeDef GPIO\_InitStructure = { 0 };

GPIO\_InitStructure.Pin = GPIO\_PIN\_All;

GPIO\_InitStructure.Mode = GPIO\_MODE\_ANALOG;

GPIO\_InitStructure.Pull = GPIO\_NOPULL;

// disable GPIO\_B

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStructure);

// disable GPIO\_B clock

\_\_HAL\_RCC\_GPIOB\_CLK\_DISABLE();

// disable GPIO\_C

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStructure);

// disable GPIO\_C clock

\_\_HAL\_RCC\_GPIOC\_CLK\_DISABLE();

}

Figure IV.3.6: source code for function disable\_gpio in main.c

The pins used in this project was chosen so that only GPIO port A is needed, therefore GPIO port B and C can be disabled.

#### Put XBee to sleep

The client XBee module is set into sleep mode 2 (ATSM 2 - pin doze). This mode consume more power but has a shorter wake up time. When in this mode, pin 9 (SLEEP\_RQ) will put the XBee to sleep if its value is high, and wake the XBee if the value is low. This pin is control by pin D9 (PA8) on the STM32

#### Put STM32 to sleep

// enter sleep mode

void enter\_sleep\_mode() {

// put XBee to sleep

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_8, GPIO\_PIN\_SET);

// disable SysTick interrupt

HAL\_SuspendTick();

// put the STM32 into sleep mode

HAL\_PWR\_EnterSLEEPMode(PWR\_MAINREGULATOR\_ON, PWR\_SLEEPENTRY\_WFI);

// after wake up

// wake the XBee

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_8, GPIO\_PIN\_RESET);

// enable Systick interrupt

HAL\_ResumeTick();

// wait for the XBee wake up time

HAL\_Delay(2);

}

Figure IV.3.7 source code for function enter\_sleep\_mode in main.c

In order to put the STM32 to sleep, the function HAL\_PWR\_EnterSLEEPMode is used. This function take two arguments:

* PWR\_MAINREGULATOR\_ON: choose the regulator in sleep mode, either the main regulator or the low power regulator. In this project, the main regulator is used for sleep mode
* PWR\_SLEEPENTRY\_WFI: choose the Cortex-M sleep instruction, either WFI or WFE. In this project, the WFI instruction is used, which will wake the STM32 both on event or on registered interrupt

All interrupt that was not intended to wake the STM32 up must also be disabled. In this case, the SysTick timer interrupt must be disable by the function HAL\_SuspendTick. If HAL\_Delay is use after wake up, this interrupt must be re-enable by the function HAL\_ResumeTick

The function enter\_sleep\_mode also contain the code to put XBee to sleep.

#### Result

After setting up the sleep mode, total current drain was measure at the connection with the battery. The results are shown below

* In sleep mode: 60*mA*
* Out of sleep mode: 110*mA*
* Wake time to send one package to the server: 120*ms*

With the battery capacity of 500*mAh*, this client can last , which is not commercially acceptable.

Further inspection review that while the XBee was put to sleep and drain a current of less than 1*mA*, the STM32 still pulling a current of more than 50*mA,* while its logical behavior is identical to being in sleep mode. Upon consulting another group with a similar solution, it seems that a similar software can put their STM32 into sleep mode with lower than 1 *mA* drain. The only difference is that while our solution use a 9*V* battery, they use a 3.3*V* battery. This may show that the ST-LINK was not properly shutdown during sleep mode in our software.

### Main function

/\* USER CODE BEGIN 2 \*/

// pull down sensor interrupt pin

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_2, GPIO\_PIN\_RESET);

// register the UART handle used to speak to XBee

init\_XBee(&huart1);

// disable unused GPIO ports

disable\_gpio();

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1) {

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

// current door status, old door status

int stt, old\_stt;

// read current door status

stt = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_2);

do {

// record old door status

old\_stt = stt;

// send XBee package depend on the door status

if (stt == GPIO\_PIN\_SET) {

tx\_req\_XBee(1, 0xAB01, 0, "2+1");

} else {

tx\_req\_XBee(1, 0xAB01, 0, "2+0");

}

// read new door status

stt = HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_2);

// only exit if the status of the door is not changed during XBee transmission

} while (stt != old\_stt);

// enter sleep mode

enter\_sleep\_mode();

/\* USER CODE BEGIN 3 \*/

}

Figure IV.3.8 key source code in main.c

In the main function, after setting up all the peripherals:

* Sensor interrupt pin is pull down to 0 in order for the interrupt to register
* Init XBee with the right UART handle
* Disable unused GPIO
* Enter the main program loop

In the loop, another loop is used to repeatedly send new update to the server if the state of the door is changed while XBee is doing transmission. This is to prevent cases when the sensor fluctuate and the last state of the door is not correctly registered to the server.

## Server side

### XBee receive frame for 16-bit address

This frame will be receive by the Raspberry Pi server when a package is receive by the sever XBee module.

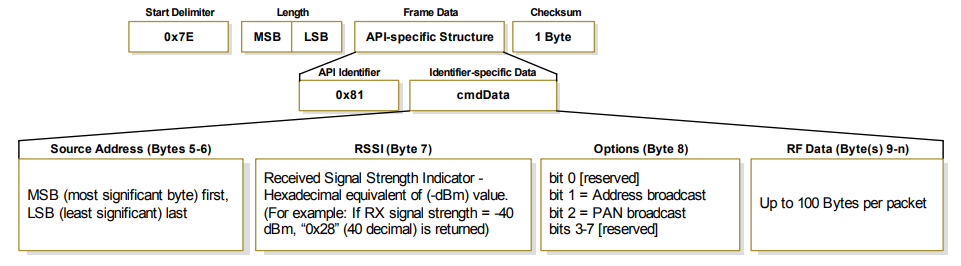


Figure IV.4.1: Layout of one receive frame for 16-bit address frame

A receive frame for 16-bit address has five fields in its frame data:

* API identifier (0x80, 1 byte): specify the API function
* Source address (2 bytes): specify the sending module’s address
* RSSI (1 byte): show the signal strength of the received package
* Option (1 byte): has two flag
  + 0x02: set if the package was sent to broadcast adress
  + 0x04: set if the package was sent with PAN broadcast ID
* RF data (maximum 100 bytes): data received

### Config.py

Config.py is a python script that can open an AT command console to the connected XBee module. Using this file, a user can modify the XBee settings without needing a dedicated connector.

#! /usr/bin/python3

import sys

import serial

import time

# define the serial port to connect to the XBee

XBee = serial.Serial('/dev/ttyS0', 9600)

# send the ‘+++’ string to enter command mode

def enter\_cmd\_mode():

print('Getting XBee into command mode ...')

XBee.write(b'+++')

count = 0

while XBee.inWaiting() < 3:

count += 1

time.sleep(0.01)

if count > 500:

break

recv = XBee.read(3)

if XBee.inWaiting() > 0 or recv != b'OK\r':

raise RuntimeError('Error getting XBee into command mode')

print('XBee is now command mode')

# send raw data to the XBee

def send\_raw(cmd):

XBee.write(cmd.encode('UTF-8'))

# receive raw data from the XBee

# return a string instead of a bytes array

def recv\_raw():

count = 0

while XBee.inWaiting() < 1:

count += 1

time.sleep(0.01)

if count > 1000:

print("Receive timeout")

enter\_cmd\_mode()

break

len = XBee.inWaiting()

time.sleep(0.01)

while XBee.inWaiting() != len:

len = XBee.inWaiting()

time.sleep(0.01)

return XBee.read(len).decode('UTF-8')

# the main function for the console

def main():

enter\_cmd\_mode()

while True:

line = sys.stdin.readline().rstrip()

send\_raw("{}\r".format(line))

reply = recv\_raw().rstrip()

print(reply)

# run only if this file is executed, not imported

if \_\_name\_\_ == '\_\_main\_\_':

try:

main()

# exit on Ctrl+C press

except KeyboardInterrupt:

print()

print('Good bye!')

Figure IV.4.2: source code in config.py

### Server.py

Config.py is a python script that start a server to receive data from the client and display it onto the screen.

#!/usr/bin/python3

import serial

from time import sleep

# define serial port to the XBee

XBee = serial.Serial ("/dev/ttyS0", 9600)

while True:

# wait till there is data

while XBee.inWaiting() < 1:

sleep(0.01)

# read one byte

c = XBee.read()

# if the byte is 0x7E

if c == b"\x7e":

# get length

buf = XBee.read(2)

len = buf[0]\*256 + buf[1]

# get API identifier

apid = int.from\_bytes(XBee.read(1), "big")

if apid == 0x81:

# get address

addr = int.from\_bytes(XBee.read(2), "big")

# get signal strength

str = int.from\_bytes(XBee.read(1), "big")

# get option

opt = int.from\_bytes(XBee.read(1), "big")

# get data

data = XBee.read(len-5).decode("UTF-8").rstrip()

# if the package come from the right address and has the right format

if addr == 0xAB02 and data[:2] == "2+":

value = data[2]

# display the door status

if value == '1':

print("Door closed")

else:

print("Door open")

Figure IV.4.3: source code in server.py

# PRICING

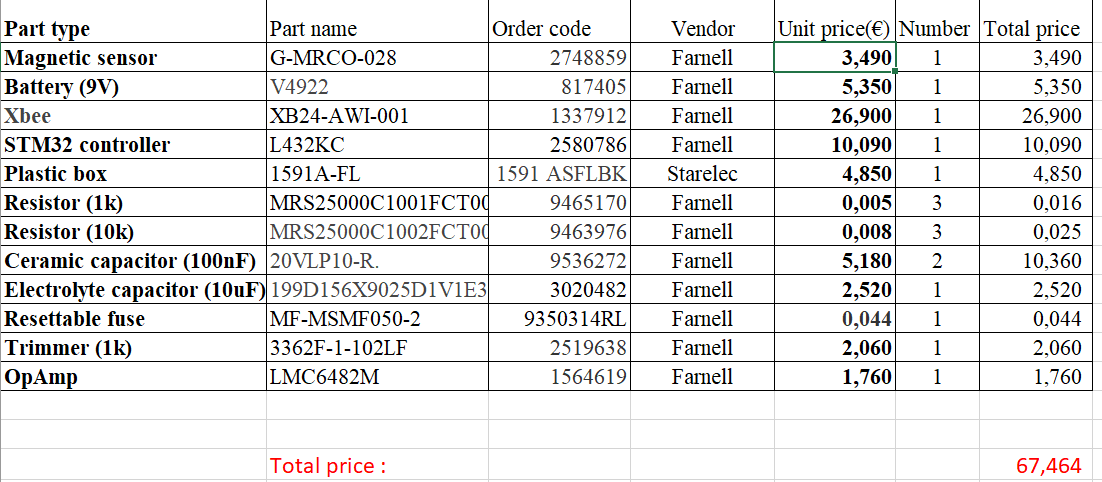


Table V.1.1: List of all hard ware devices

To overcome the challenge of ensuring the functionality of the device with such constraint on size and components, great consideration was taken by us. We calculated the size of the device and check datasheet of every single items to ensure that the components and battery not only fit in 45.8mm x 95.8mm x 19.0mm box.

The table above show the names, order codes and price of all implements. The actual selling price still have not appraised at the moment but the price between 150 and 200 euro should be good for develop team. Hence, taking 80€ to 130€ benefit per sale is a big motivation for the company to continue do product research and development. In the other hand, the company also committed to providing software support and guarantee security to customers, make them feel worthy of the money.

# REFERENCE

Farnell AN AVNET COMPANY. (2019, April 28). *NUCLEO-L432KC - Development Board, STM32L432KC MCU, ST-LINK/V2-1 Debugger/Programmer, Arduino Connectivity.* Retrieved from Farnell AN AVNET COMPANY: <https://fi.farnell.com/stmicroelectronics/nucleo-l432kc/dev-board-nucleo-32/dp/2580786?ost=L432KC&ddkey=https%3Afi-FI%2FElement14_Finland%2Fsearch>

Mouser Electronics. (2019, April 28). G-MRCO-028 Measurement Specialties | Mouser Europe. Retrieved from Mouser Electronics: <https://eu.mouser.com/ProductDetail/Measurement-Specialties/G-MRCO-028?qs=oFx6pF86PmDibzYzasua3w==>

Wikipedia, the free encyclopedia. (2019, April 28). Wheatstone bridge - Wikipedia. Retrieved from Wikipedia, the free encyclopedia: <https://en.wikipedia.org/wiki/Wheatstone_bridge>

Texas Instruments Incorporated. (2015). LMC6482 CMOS Dual Rail-to-Rail Input and Output Operational Amplifier. Retrieved from ti.com:<https://www.ti.com/lit/ds/symlink/lmc6482.pdf>

Digi Internatonal, Inc. (2009). XBee®/XBee-PRO® RF Modules. Retrieved from sparkfun.com: <https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>

STMicroelectronics (2019). STM32CubeMX for STM32 configuration and initialization C code generation. Retrieved from st.com: <https://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/translations/en.DM00104712.pdf>

STMicroelectronics (2018). Reference manual STM32L4x5 and STM32L4x6 advanced Arm®-based 32-bit MCUs. Retrieved from st.com: <https://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf>

STMicroelectronics (2018). User manual STM32 Nucleo-32 boards (MB1180). Retrieved from st.com: <https://www.st.com/content/ccc/resource/technical/document/user_manual/e3/0e/88/05/e8/74/43/a0/DM00231744.pdf/files/DM00231744.pdf/jcr:content/translations/en.DM00231744.pdf>