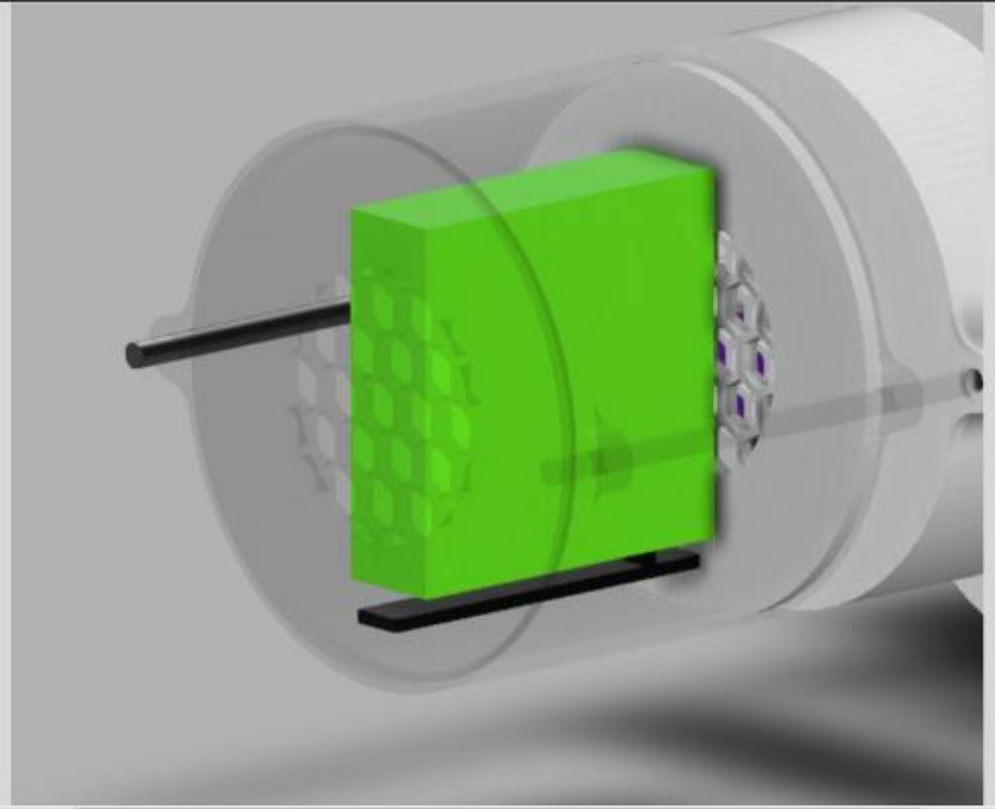
Particolato Condiviso

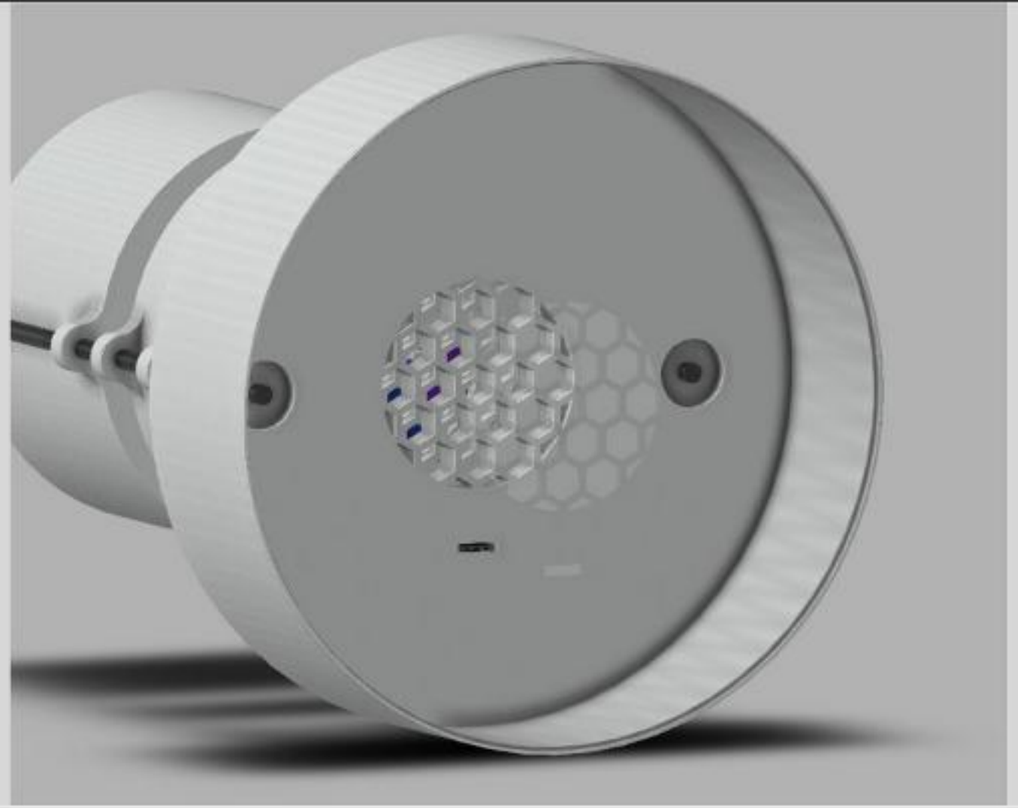
What

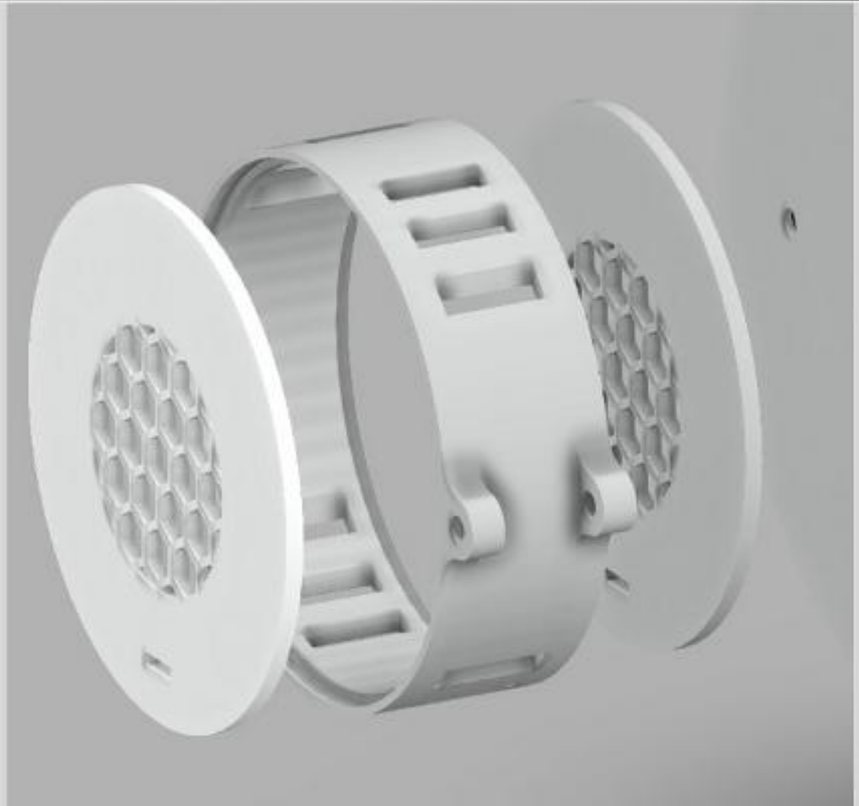
What

- Progetto sviluppato in collaborazione con CNR e 5F
- Mirato alla raccolta di dati sul particolato
- Condividere con rete scolastica ed europea

Hardware







Adobe Fusion

CENTRAL_CONTROLLER
ESP32-WROVER-B

+3V
GND
GND
GND
GND
GND
VCC
+3V

1 3v3
2 RST
3 36
4 39
5 34
6 35
7 32
8 33
9 25
10 26
11 27
12 14
13 12
14 13
15 GND
16 VBat
17 GND

GND 34
23 33
22 32 SCL
TX0 / D1 31 ESP_TX
RX0 / D3 30 ESP_RX
21 29 SDA
19 28
18 27
5 26
4 25
0 24
2 23
15 22
GND 21
5V 20
VIN 19
3V3 18

BAT
USB
SD

USB
CJMCU Micro USB Breakout

GND
VCC

5 GND 5
4 ID 4
3 D+ 3
2 D- 2
1 VCC 1

GSM
SIM800L

VCC
ESP_RX
ESP_TX
GND

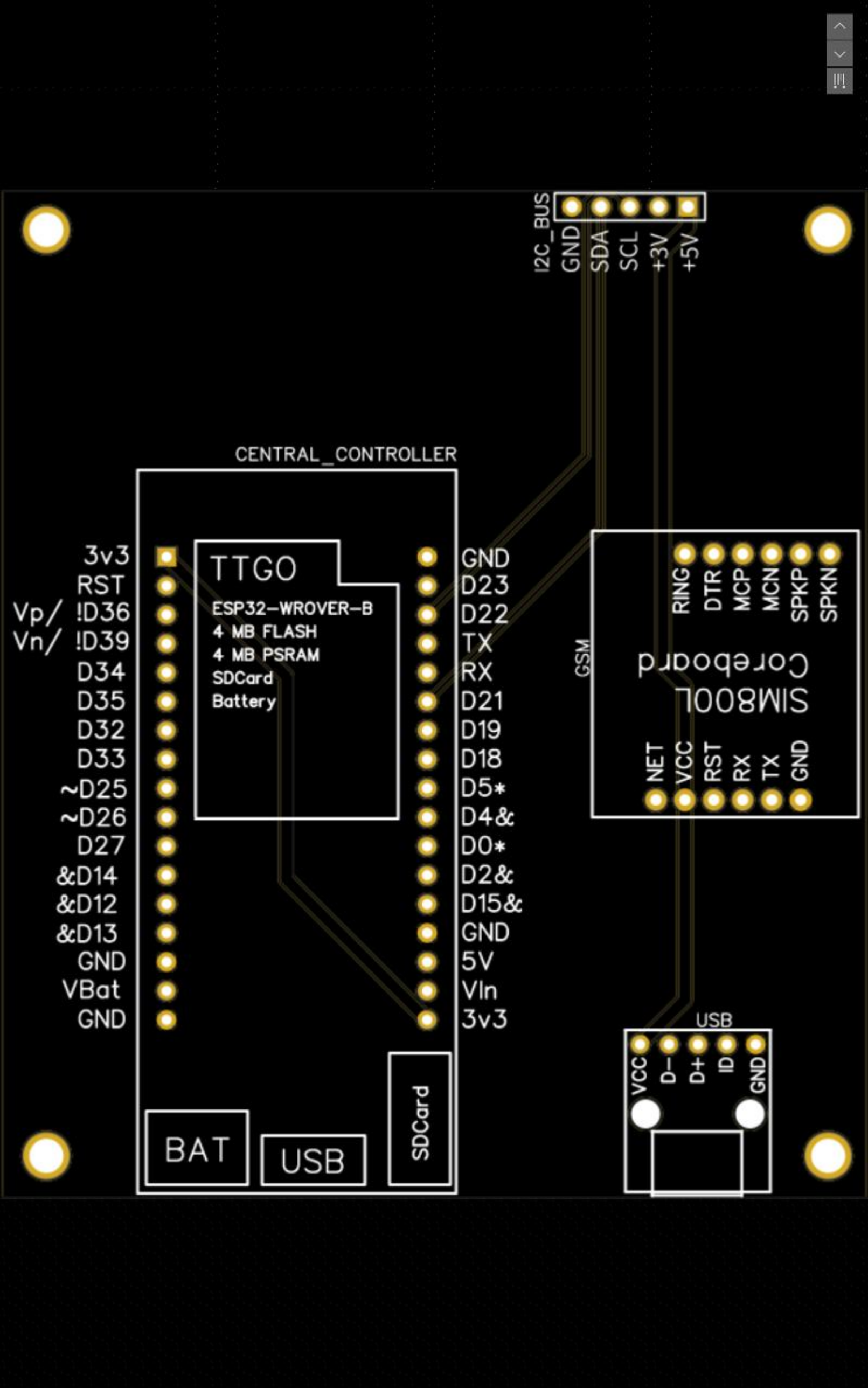
NET SIM800L RING
VCC DTR
RST MCP
RXD MCN
TXD SPKP
GND SPKN

GND

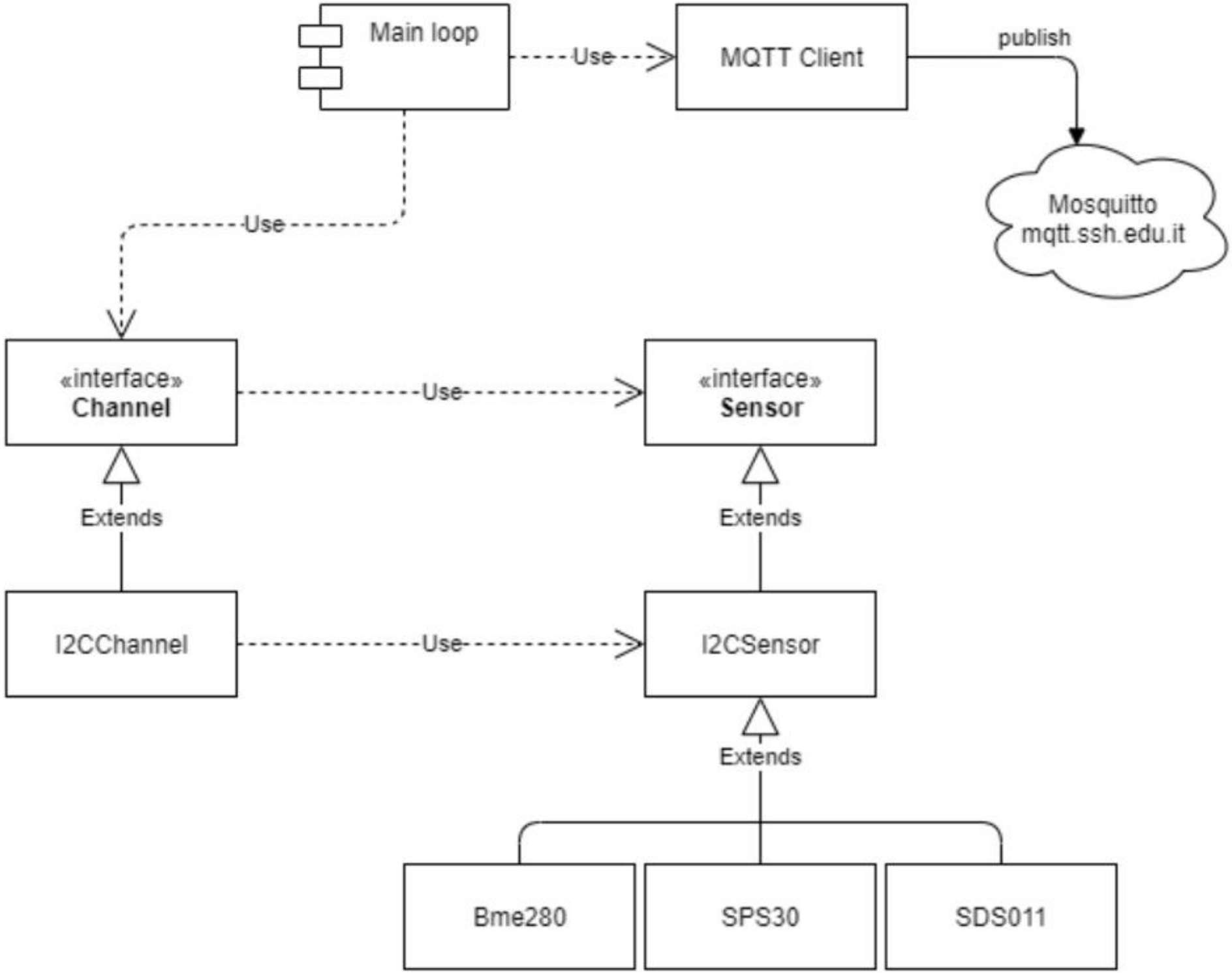
I2C_BUS
VCC ESQ-105-12-T-S

+3V
SCL
SDA
GND

1
2
3
4
5


I2C_BUS
GND
SDA
SCL
+3V
+5V
CENTRAL_CONTROLLER
TTGO
ESP32-WROVER-B
4 MB FLASH
4 MB PSRAM
SDCard
Battery
3v3
RST
Vp/ !D36
Vn/ !D39
D34
D35
D32
D33
~D25
~D26
D27
&D14
&D12
&D13
GND
VBat
GND
GND
D23
D22
TX
RX
D21
D19
D18
D5*
D4&
D0*
D2&
D15&
GND
5V
Vin
3v3
GSM
SIM800L
Coreboard
RING
DTR
MCP
MCN
SPKP
SPKN
NET
VCC
RST
RX
TX
GND
USB
VCC
D-
D+
ID
GND
BAT
USB
SDCard

Software


Main loop
Use
MQTT Client
publish
Mosquitto
mqtt.ssh.edu.it
Use
«interface»
Channel
Use
«interface»
Sensor
Extends
Extends
I2CChannel
Use
I2CSensor
Extends
Bme280
SPS30
SDS011

Strong code

```
template <typename SensorType>
class Channel
{
public:
    virtual string readNext() = 0;

    void addSensor   (std::unique_ptr<SensorType>&& sensor);
    void removeSensor(const byte id);

    void initAll() const;

protected:
    Channel(std::vector<std::unique_ptr<SensorType>> sensors);
    Channel() : Channel({}) {}

    SensorType* getNext();

    std::vector<
        std::unique_ptr<SensorType>
    > _sensors;

public:
    Channel(const Channel&&) = delete;
    Channel(Channel&)        = delete;
};
```

```cpp
template <typename SensorType>
Channel<SensorType>::Channel(/* ... */)
{
    static_assert(
        std::is_base_of<sensors::Sensor, SensorType>::value,
        "`SensorType` must be derived from `Sensor`");
}
```

```
class I2CChannel : public Channel<sensors::I2CSensor>
{
public:
    I2CChannel(std::vector<std::unique_ptr<sensors::I2CSensor>> sensors)
        : Channel(move(sensors)) {}

    I2CChannel() : Channel({}){}
    string readNext();
};

string I2CChannel::readNext()
{
    const auto sensor = this->getNext();

    auto data = sensor->read();
    return sensor->decode(data);
}
```

```
struct SensorData
{
    byte id;
    std::vector<byte> data;
};

class Sensor
{
public:
    virtual string decode(const SensorData& rawData) const = 0;
    virtual void   init() = 0;

    const byte   id;
    const string name;

protected:
    Sensor(
        const byte   id,
        const string name
    ) :
        id(id),
        name(name)
    {};
};
```

```
class I2CSensor : public Sensor
{
public:
    virtual SensorData read() const = 0;

protected:
    I2CSensor(
        const byte id,
        const string name
    ) : Sensor(id, name) {}
};
```

Finally, a sensor

```
class Bme280 : public I2CSensor
{
public:
    Bme280();

    void init();
    byte syncId() const;

    auto read()                    const -> SensorData;
    auto decode(const SensorData&) const -> string;

    int32_t computeTemperature(dword rawTemperature) const;
    dword   computePressure   (dword rawPressure)    const;
    dword   computeHumidity   (dword rawHumidity)    const;

    static dword computeRawTemperature(const std::vector<byte>& data);
    static dword computeRawPressure   (const std::vector<byte>& data);
    static dword computeRawHumidity   (const std::vector<byte>& data);

    static auto sdtocb(const std::vector<byte>& data) -> Bme280CalibrationParams;

private:
    Bme280CalibrationParams _calibrationParams;

    auto readCalibrationParams() -> Bme280CalibrationParams;
};
```

```cpp
Bme280::Bme280() : I2CSensor(0x76, "bme280") {}


void Bme280::init()
{
    i2c::write<6>(this->id,
    {
        Bme280Register::CONFIG,
        0xA0,   // Sampling rate 1 sec

        Bme280Register::CTRL_HUM,
        0x01,

        Bme280Register::CTRL_MEAS,
        0x27,   // Pressure and temperature data acquisition options
    });

    this->_calibrationParams = readCalibrationParams();
}
```

```
SensorData Bme280::read() const
{
    constexpr byte TOTAL_BYTES = 8;
    SensorData res {
        .id   = this->id,
        .data = vector<byte>(TOTAL_BYTES),
    };

    i2c::beginReadTransaction(
        this->id,
        Bme280Register::ADDR_DATA_LOW,
        TOTAL_BYTES);

    i2c::readMultiple<byte, TOTAL_BYTES>(res.data);
    return res;
}
```

```
string Bme280::decode(const SensorData& rawData) const
{
    StaticJsonDocument<JSON_OBJECT_SIZE(4)> doc;

    const auto& data = rawData.data;
    const dword rawTemperature = computeRawTemperature(data);
    const dword rawPressure    = computeRawPressure(data);
    const dword rawHumidity    = computeRawHumidity(data);

    doc["id"]          = rawData.id;
    doc["temperature"] = computeTemperature(rawTemperature);
    doc["pressure"]    = computePressure(rawPressure);
    doc["humidity"]    = computeHumidity(rawHumidity);

    char output[56];
    serializeJson(doc, output);
    return string(output);
}
```

```
I2CChannel i2c;
AsyncMqttClient mqttClient;

void setup()
{
    i2c.addSensor(cc::make_unique<Bme280>());
    i2c.initAll();

    WiFi.begin(WIFI_SSID, WIFI_PASS);

    mqttClient.setServer(MQTT_HOSTNAME, MQTT_PORT);
    mqttClient.connect();
}

void loop()
{
    const auto data = i2c.readNext();
    mqttClient.publish(
        WEATHER_DATA_TOPIC, 0, false, data.c_str());

    delay(1000);
}
```

```
I2CChannel i2c;
AsyncMqttClient mqttClient;
std::mutex mqttLock;

void setup()
{
    i2c.addSensor(cc::make_unique<Bme280>());
    i2c.initAll();

    WiFi.begin(WIFI_SSID, WIFI_PASS);

    mqttClient.setServer(MQTT_HOSTNAME, MQTT_PORT);
    mqttClient.connect();

    const auto readingThread = std::thread([]
    {
        const auto data = i2c.readNext();

        { std::lock_guard<std::mutex>(mqttLock);
            mqttClient.publish(
                WEATHER_DATA_TOPIC, 0, false, data.c_str());
        }
        delay(1000);
    });
}
```

Grazie per l'attenzione

Di Adriano Buffagni, Stefano Calabretti

Con l'aiuto di Matteo Budriesi e Riccardo Rosi

Github repository

Consiglio Nazionale delle Ricerche

Di Adriano Buffagni, Stefano Calabretti

Con l'aiuto di Matteo Budriesi e Riccardo Rosi

Github repository

Elettronica

Chimica

Informatica

I.T.I.S. Enrico Fermi