Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення)

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

# Пояснювальна записка

до дипломної роботи
другий (магістерський)
(рівень вищої освіти)

на тему: "Розробка апаратно-програмних засобів моніторингу метеопараметрів з використанням технологій Internet of Things (IoT)"

Виконав: студент VI курсу групи КН-61М
спеціальності 122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Кубрак О. Я.
(прізвище та ініціали)

Керівник Шабатура Ю. В.
(прізвище та ініціали)

Рецензент Щербовських С. В.
(прізвище та ініціали)

Львів – 2021 року

Національний лісотехнічний університет України
( повне найменування вищого навчального закладу )

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Галузь знань 12 "Інформаційні технології"
Спеціальність 122 "Комп'ютерні науки"
Рівень вищої освіти другий (магістерський)

**ЗАТВЕРДЖУЮ**
**завідувач кафедри**
_____Крошний І.М.
"____" _____2021 року

# З А В Д А Н Н Я
## НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

*Кубраку Олегу Ярославовичу*
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи *Розробка апаратно-програмних засобів моніторингу метеопараметрів з використанням технологій Internet of Things (IoT)*
керівник роботи *Шабатура Юрій Васильович, д.т.н., професор, професор кафедри ІТ*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *31 грудня 2020 року № С-593*
2. Термін подання студентом проекту (роботи) *10 грудня 2021 року*
3. Вихідні дані до проекту (роботи) *Спроектувати апаратне забезпечення IoT - системи моніторингу метеопараметрів на платі Arduino Mega2560, цифровому сенсорі температури, вологості і тиску BME280, алфавітно-цифровому рідкокристалічному дисплеї 16×2 на базі контролера HD44780, Wi-Fi модулі ESP-01 на чіпі ESP8266, 4 двохконтактних кнопках на 4 виводи. Створити модель IoT - пристрою моніторингу метеопараметрів в САПР Proteus VSM. Розробити програмні модулі для роботи з сенсором BME280, W-Fi модулем ESP-01 на чіпі ESP8266 та основне програмне забезпечення збору та обробки даних для МК плати Arduino Mega2560. Розробити програмне забезпечення для комунікації по протоколу MQTT з IoT – платформою ThingsBoard та сервером Mosquitto. Створити панелі візуалізації метеоданих на платформі ThingsBoard та з MQTT – брокера Mosquitto засобами Node-RED.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
*Вступ*
*Розділ І. Аналіз та дослідження проблемної області*
*Розділ ІІ. Інструментальні засоби розробки IoT - системи моніторингу метеопараметрів*
*Розділ ІІІ. Апаратне забезпечення IoT - системи моніторингу метеопараметрів*
*Розділ IV. Програмно - алгоритмічне забезпечення IoT - системи моніторингу метеопараметрів*

*Розділ V. Моделювання та дослідження макету IoT - системи моніторингу метеопараметрів*

*Висновки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

*загальна структура IoT - системи моніторингу метеопараметрів, електрична принципова схема апаратного забезпечення IoT - системи моніторингу метеопараметрів спроектована в САПР Proteus VSM, алгоритм роботи IoT - системи моніторингу метеопараметрів, алгоритм ініціалізації LCD, сенсора BME280, Wi-Fi модуля ESP-01, програмна реалізація алгоритму роботи IoT – системи моніторингу метеопараметрів, моделювання та дослідження макету IoT - системи моніторингу метеопараметрів.*

6. Консультанти по роботі, із зазначенням розділів проекту, що стосується їх

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| I-V | Шабатура Ю. В., професор кафедри ІТ | | |
| | | | |

7. Дата видачі завдання *18 грудня 2020 року*

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Огляд літератури та інтернет-ресурсів за вказаною тематикою. | 15.06.2021 р. 10.07.2021 р. | виконано |
| 2 | Постановка задачі та вибір елементної бази для проектування IoT – системи моніторингу метеопараметрів. | 11.07.2021 р. 20.07.2021 р. | виконано |
| 3 | Проектування АЗ IoT – системи моніторингу метеопараметрів засобами САПР Proteus VSM. | 12.07.2021 р. 18.08.2021 р. | виконано |
| 4 | Розроблення алгоритму роботи IoT – системи моніторингу метеопараметрів. | 19.08.2021 р. 24.08.2021 р. | виконано |
| 5 | Розроблення програмних бібліотек для роботи з цифровим сенсором температури, вологості і тиску BME280, Wi-Fi модулем ESP-01 на чіпі ESP8266. | 25.08.2021 р. 26.09.2021 р. | виконано |
| 6 | Програмна реалізація головного алгоритму роботи IoT – системи моніторингу метеопараметрів. | 27.09.2021 р. 28.10.2021 р. | виконано |
| 7 | Моделювання, створення макету та тестування IoT – системи моніторингу метеопараметрів. | 29.10.2021 р. 14.11.2021 р. | виконано |
| 8 | Оформлення записки до дипломної роботи. | 15.11.2021 р. 03.12.2021 р. | виконано |
| 9 | Здача пояснювальної записки на рецензування. | 04.12.2021 р. 10.12.2021 р. | виконано |
| 10 | Підготовка доповіді та мультимедійної презентації. | 15.12.2021 р. 17.12.2021 р. | виконано |

**Студент** _____ Кубрак О. Я.
( підпис ) (прізвище та ініціали)

**Керівник роботи** _____ Шабатура Ю. В.
( підпис ) (прізвище та ініціали)

# РЕФЕРАТ

Дипломна робота містить 118 сторінок пояснювальної записки, 35 рисунків, 3 таблиці, 3 додатки, 16 джерел.

У магістерській дипломній роботі розроблено апаратно-програмні засоби моніторингу метеопараметрів з використанням технологій Internet of Things (IoT). Розроблено IoT – пристрій, який моніторить такі метеопараметри як: температура, відносна вологість і атмосферний тиск. Апаратне забезпечення IoT – пристрою моніторингу метеопараметрів складається з мікроконтролерного пристрою, який побудовано на платі Arduino Mega2560, цифровому сенсорі температури, вологості і тиску BME280, Wi-Fi модулі ESP-01 на чіпі ESP8266 та символьному рідкокристалічному дисплеї 16×2 на контролері Hitachi HD44780. Розроблено електричну принципову схему та модель мікроконтролерного пристрою моніторингу метеопараметрів в САПР Proteus VSM. Розроблено алгоритм роботи IoT - системи моніторингу метеопараметрів. Розроблено програмні модулі для роботи з сенсором BME280 та Wi-Fi модулем ESP-01 та основне програмне забезпечення збору та обробки даних для МК плати Arduino Mega2560 на мові С в середовищі Arduino IDE. Створено програмне забезпечення для комунікації по протоколу MQTT з IoT – платформою ThingsBoard та сервером Mosquitto. Створено панелі візуалізації метеоданих на IoT - платформі ThingsBoard засобами ThingsBoard Dashboard, а з локального MQTT – брокера Mosquitto засобами Node-RED.

Змодельовано роботу IoT - пристрою моніторингу метеопараметрів в емуляторі Proteus ISIS. Зібрано макет мікроконтролерного пристрою моніторингу метеопараметрів та протестовано його роботу.

**Ключові слова:** IoT - пристрій, моніторинг метеопараметрів, плата Arduino Mega2560, цифровий сенсор тиску, температури і вологості BME280, Wi-Fi модуль ESP-01 на чіпі ESP8266, LCD HD44780, САПР Proteus VSM, мова програмування С, вбудоване ПЗ, середовище для написання і завантаження програмного коду в МК плати Arduino - Arduino IDE, протокол MQTT, брокер (сервер) Mosquitto, Node-RED, відкрита IoT - платформа ThingsBoard.

# ABSTRACT

The master's thesis contains 118 pages of explanatory note, 35 pictures, 3 tables, 3 applications, 16 used literature sources.

In the master's thesis, the hardware and software of the IoT based weather monitoring device have been developed. The IoT device monitors such weather parameters as: atmospheric pressure, temperature and relative humidity.

The hardware of the IoT based weather monitoring device consists of the microcontroller device which is developed on the Arduino Mega2560 board, digital pressure, temperature and humidity sensor BME280, Wi-Fi module ESP-01 built on the ESP8266 chip and the alpha-numeric LCD 16×2 based on the Hitachi HD44780 controller.

The electronic circuit and the model of the microcontroller weather monitoring device have been created in Proteus Design Suite. The operation algorithm of the IoT based weather monitoring system has been developed.

The software modules for communication with the BME280 sensor and Wi-Fi module ESP-01 and main data collection and acquisition software for the microcontroller of the Arduino Mega2560 board have been created in C using the Arduino IDE. The software for communication with the IoT platform ThingsBoard and MQTT broker Mosquitto has been created. The dashboards for weather data visualization on the IoT platform ThingsBoard using ThingsBoard Dashboard and from the local MQTT broker Mosquitto using Node-RED have been created.

The operation of the IoT based weather monitoring device has been simulated in Proteus ISIS. The prototype of the IoT device for weather monitoring has been built on a solderless breadboard and its operation has been tested.

**Key words:** IoT device, weather monitoring, Arduino Mega2560 board, digital pressure, humidity and temperature sensor BME280, Wi-Fi module ESP8266 ESP-01, alpha-numeric LCD HD44780, ECAD system Proteus VSM, embedded C, embedded software, open-source software for writing and uploading code to the Arduino board - Arduino IDE, MQTT protocol, Mosquitto broker, Node-RED, open-source IoT platform ThingsBoard.

## ТЕХНІЧНЕ ЗАВДАННЯ

Розробити апаратно-програмні засоби моніторингу метеопараметрів з використанням технологій Internet of Things (IoT). Апаратне забезпечення IoT – пристрою моніторингу метеопараметрів розробити з використанням наступних електронних комплектуючих:

- Плата Arduino Mega2560 на мікроконтролері AVR ATmega2560.
- Символьний рідкокристалічний дисплей 16×2 на контролері HD44780.
- Цифровий сенсор тиску, температури і вологості BME280.
- Wi-Fi модуль ESP-01 на чіпі ESP8266.
- Модуль живлення та узгоджувач рівнів 5 В – 3,3 В.
- 4 двохконтактні кнопки на 4 виводи.

1. Спроектувати IoT - пристрій системи моніторингу метеопараметрів засобами САПР Proteus.
2. Розробити алгоритм роботи IoT - пристрою моніторингу метеопараметрів.
3. Створити програмний модуль роботи з цифровим сенсором BME280.
4. Створити програмний модуль роботи з Wi-Fi модулем ESP-01.
5. Створити програмний клієнт мікроконтролерного пристрою для обміну даними з IoT – сервісами та серверами, що використовують протокол MQTT.
6. Створити програмне забезпечення мікроконтролерного пристрою для обміну даними з відкритою IoT - платформою ThingsBoard.
7. Створити програмний клієнт мікроконтролерного пристрою для обміну даними з MQTT – брокером Mosquitto.
8. Створити засобами Node-RED IoT панель візуалізації метеоданих отриманих з MQTT – брокера Mosquitto.
9. Розробити вбудоване програмне забезпечення IoT - пристрою моніторингу метеопараметрів відповідно до алгоритму його роботи.
10. Провести моделювання та дослідження макету IoT - пристрою моніторингу метеопараметрів.

# ЗМІСТ

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

**ПК** – персональний комп’ютер;

**АЗ** – апаратне забезпечення;

**МК** – мікроконтролер;

**ЕК** – електрична схема;

**ПЗ** – програмне забезпечення;

**IoT - Internet of Things (Інтернет речей)** це мережа пов’язаних через Інтернет об’єктів, здатних збирати дані і обмінюватися ними, які надходять з вбудованих сервісів. Пристрої, що входять до Інтернету речей (IoT) – це будь-які автономні пристрої, підключені до Інтернету, які можуть відслідковуватися і/або управлятися дистанційно.

**MQTT** (англ. Message Queuing Telemetry Transport) – спрощений мережевий протокол, що працює поверх TCP/IP, орієнтований для обміну повідомленнями між пристроями за принципом видавець–підписник. Протокол MQTT призначений для передачі послідовності повідомлень з телеметричними даними, тобто інформації від сенсорів температури, вологості, освітленості та інших. Протокол MQTT використовується для обміну повідомленнями в Інтернеті речей (Internet of Things (IoT);

**САПР (CAD)** – система автоматизованого проектування;

**Arduino Mega2560** – апаратно-програмна платформа на МК AVR ATmega2560;

**Arduino IDE** – інтегроване середовище для написання і прошивки програмного коду в МК плати Arduino;

**AVR** – сімейство 8-бітних мікроконтролерів фірми Atmel;

**FLASH** – різновид напівпровідникової технології пам’яті, яка перепрограмовується електричним методом;

**EEPROM** – (англ. Electrically Erasable Programmable Read-Only Memory) – постійна пам’ять, яка стирається і перепрограмовується електричним методом;

**ПЗП (ROM)** – (англ. read-only memory) постійний запам’ятовуючий пристрій. Незалежна пам’ять, використовується для зберігання масиву незмінних даних;

**ОЗП (RAM)** – оперативна пам’ять або оперативний запам’ятовуючий пристрій (англ. Random Access Memory, RAM);

**АЦП (ADC)** – аналого-цифровий перетворювач;

**UART** – (Universal Asynchronous Receiver/Transmitter) універсальний асинхронний приймач та передавач, УАПП;

**SPI** (Serial Peripheral Interface, SPI bus) – послідовний периферійний інтерфейс, шина SPI) – послідовний синхронний повнодуплексний стандарт передачі даних, розроблений фірмою Motorola для забезпечення простого підключення мікроконтролера та периферії;

**$I^2C$** – послідовна шина даних для зв’язку інтегральних схем, що використовує дві двонаправлені лінії зв’язку (SDA (лінія даних) і SCL (лінія синхронізації));

**РКД (LCD)** – рідкокристалічний дисплей;

**ШІМ (PWM)** - широтно-імпульсна модуляція (pulse-width modulation), це операція отримання змінного в часі аналогового сигналу за допомогою цифрових пристроїв. Пристрої використовуються для отримання прямокутних імпульсів-сигналу, який постійно переключається між максимальним і мінімальним значеннями. Даний сигнал моделює напругу між максимальним значенням (5 В) і мінімальним (0 В), змінюючи при цьому тривалість часу включення 5 В відносно включення 0 В. Тривалість включення максимального значення називається шириною імпульсу. Для отримання різних аналогових величин змінюється ширина імпульсу.

# ВСТУП

Розвиток електроніки, технологій програмування та Інтернет привів до появи Інтернету речей. Інтернет речей (Internet of Things (IoT)) – це комплекс технологій для збору інформації з системи розподілених сенсорів і дистанційного керування пристроями, які підключені до мережі Інтернет, а також для зберігання, обробки і візуалізації даних на локальних або віддалених серверах. Зокрема, таким пристроєм може бути IoT - пристрій моніторингу параметрів погоди.

**Метою даної роботи** є забезпечення безперервного контролю значень основних метеопараметрів у зоні інтересів споживача, їх обробки, передачі та візуалізації, на основі використання апаратної платформи Arduino з набором відповідних сенсорів і засобів комунікації та відповідного програмного забезпечення, що дозволить отримати дешевий і одночасно практично професійний апаратно-програмний комплекс моніторингу і прогнозування погодних змін в інтересах приватного споживача.

Для розробки IoT - пристрою моніторингу метеопараметрів запропоновано використати плату Arduino Mega2560 з МК ATmega2560, цифровий сенсор атмосферного тиску, температури і вологості BME280, Wi-Fi модуль ESP-01 на чіпі ESP8266 та символьний РК-дисплей 16×2 на контролері Hitachi HD44780.

Мікроконтролерний пристрій має вимірювати параметри погоди та передавати їх по мережі Wi-Fi з використанням протоколу MQTT на локальний сервер Mosquitto або IoT - платформу ThingsBoard. ThingsBoard зберігає, обробляє отримані IoT - дані та виводить їх на розробленій панелі візуалізації метеоданих ThingsBoard Dashboard. Сервер Mosquitto передає метеодані локальному MQTT - клієнту. Локальний MQTT-клієнт відображає їх на веб-панелі для моніторингу та візуалізації метеоданих розробленій засобами Node-RED.

На мікроконтролерах створено багато сучасних електронних пристроїв. Важливою їх особливістю для проектувальника, є те, що вони дозволяють легше та дешевше реалізувати алгоритм роботи пристрою і зменшити його габарити. За

допомогою МК можна приймати дані від різних давачів та управляти різними виконавчими пристроями (моторами, нагрівачами, освітлювальними пристроями, сервоприводами і т.п.), виконувати математичні обчислення.

**Об'єктом дослідження** є процес моніторингу метеопараметрів з використанням технологій IoT.

**Предмет дослідження** – апаратно-програмні засоби моніторингу метеопараметрів.

**Наукова новизна роботи полягає у:**

- ✓ розробленні моделі IoT - пристрою моніторингу метеопараметрів та методів (алгоритмів) опрацювання, аналізу та візуалізації виміряних метеовеличин;
- ✓ використанні нових технологій у розробленні апаратно-програмних засобів моніторингу метеопараметрів.

**Практичне значення одержаних результатів полягає у:**

- ✓ розробленні макету IoT - пристрою моніторингу метеопараметрів на сучасних недорогих та доступних комплектуючих;
- ✓ розробленні програмно-алгоритмічного забезпечення IoT – пристрою моніторингу метеопараметрів з використанням відкритих бібліотек та інструментального програмного забезпечення, що дозволяє в подальшому покращувати або розширювати його функціональні можливості;
- ✓ можливості віддаленого моніторингу метеопараметрів в реальному часі з використанням IoT – платформи.

# РОЗДІЛ І. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРОБЛЕМНОЇ ОБЛАСТІ

## 1.1. Концепція “Інтернет речей (Internet of Things)”

Інтернет речей (англ. Internet of Things, IoT) – це концепція обчислювальної мережі фізичних предметів (“речей”), що мають вбудовані технології для взаємодії один з одним або із зовнішнім середовищем. IoT відноситься до підключення пристроїв (крім звичайних комп’ютерів і смартфонів) через Інтернет. Автомобілі, кухонна побутова техніка та навіть кардіомонітори можуть бути підключені через IoT. Інтернет речей в наступні декілька років буде тільки зростати, тому в цьому списку буде з’являтися все більше пристроїв.

IoT, або Інтернет речей - це мережа зв’язаних через Інтернет об’єктів, які здатні збирати дані та обмінюватися даними, що надходять з вбудованих сервісів. Пристрої, які входять до Інтернет речей – це будь-які автономні пристрої, які підключені до Інтернету та можуть бути відслідковані і/або керовані дистанційно.

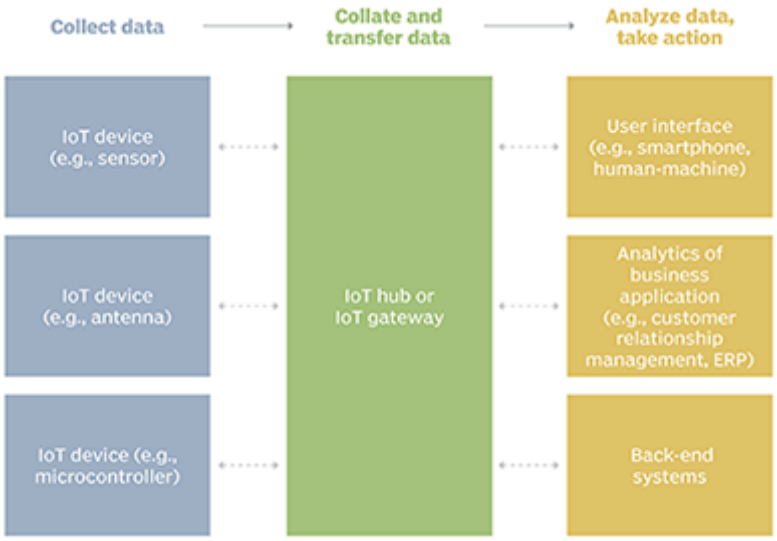



Рис.1.1. Приклад IoT – системи

Екосистема IoT (або Інтернету речей) – це всі компоненти, які дозволяють бізнесу, установам (організаціям) і користувачам підключати свої пристрої IoT, включаючи пульти управління, панелі інструментів, мережі, шлюзи, аналітику, зберігання даних і безпеку.

**Фізичний рівень** – це апаратне забезпечення, яке використовується в IoT-пристроях, включаючи сенсори та мережеве обладнання.

**Мережевий рівень** відповідає за передачу даних, зібраних на фізичному шарі (рівні), до різних пристроїв.

**Програмний рівень** включає протоколи та інтерфейси, які пристрої використовують для ідентифікації та зв'язку один з одним.

**Пульти керування** дозволяють людям використовувати IoT-пристрої, з'єднуючись з ними і контролювати їх за допомогою панелі інструментів. Наприклад, такої як мобільна аплікація (мобільна програма на мобільному пристрої (смартфоні, планшеті)) або web-сервіс. До пультів управління відносяться смартфони, планшети, ПК, розумні годинники, телевізори та нетрадиційні пульти.

**Панелі інструментів** забезпечують відображення інформації про екосистему IoT для користувачів, що дозволяє їм управляти екосистемою IoT. Зазвичай використовується віддалене управління.

**Аналітика** – програмні системи, які аналізують дані, отримані від IoT-пристроїв. Аналітика використовується у великій кількості сценаріїв, наприклад для прогнозування технічного обслуговування.

**Зберігання даних** – місце зберігання даних з IoT-пристроїв.

**Мережі** – рівень Інтернет комунікацій, який дозволяє операторам спілкуватися з пристроєм, а пристроям – спілкуватися (обмінюватися інформацією) один з одним.

Інтернет речей охоплює такі комплекси знань і вмінь як: цифрова електроніка, програмування мікроконтролерів, передача даних і протоколи мережі Інтернет, веб-дизайн та серверне веб-програмування.

Використання IoT-пристроїв є перспективним в наступних напрямках: виробництво, транспорт, оборона, сільське господарство, інфраструктура, роздрібні продажі, логістика, банки, нафта, газ, видобуток корисних копалин, страхова справа, розумні будинки, виробництво продуктів харчування, обслуговування, госпіталі, охорона здоров'я, розумні споруди, IoT-компанії.

Один пристрій IoT з'єднується з іншим для передачі інформації через Інтернет-протоколи. IoT-платформи служать мостом між сенсорами пристроїв і

мережею передачі даних. На сьогодні є велике різноманіття платформ IoT, наприклад такі як: Amazon Web Services, Microsoft Azure, ThingWorx IoT Platform, IBM's Watson, Cisco IoT Cloud Connect, Salesforce IoT Cloud, Oracle Integrated Cloud, GE Predix, Node-RED, ThingsBoard, Adafruit IO, Kaa IoT Platform, thinger.io.

## 1.2 Протокол передачі повідомлень з телеметричними даними MQTT

MQTT – це легкий протокол обміну повідомленнями publish/subscribe (публікація/підписка (видавець/підписник)), який розроблений для телеметрії M2M (від машини до машини) в середовищах з низькою пропускною здатністю. Він був розроблений Andy Stanford-Clark (IBM) і Arlen Nipper в 1999 році для підключення телеметричних систем нафтопроводів через супутник. Не дивлячись на те, що він починався як закритий протокол, він був випущений в 2010 році і став стандартом в 2014 році.

MQTT розшифровується як Message Queuing Telemetry Transport, але раніше він називався Telemetry Transport з чергою повідомлень. На сьогоднішній день MQTT є основним протоколом, що використовується для Інтернету речей (Internet of Things (IoT)).

Версії протоколу MQTT:

Є два різних варіанти MQTT та декілька версій.

- MQTT v3.1.0 –
- MQTT v3.1.1 – в загальному використанні;
- MQTT v5 – на даний час в обмеженому використанні.

Оригінальній MQTT, який розроблено в 1999 році, і є у використанні протягом багатьох років і призначений для мереж TCP/IP. Загальновживаною версією протоколу є MQTT v3.1.1. MQTT-SN визначено (описано специфікацію) приблизно в 2013 році і створено для роботи з UDP, ZigBee та іншими.

Оскільки MQTT-клієнти не мають таких адресів, як адреса електронної пошти, номер телефону і т.д., тому непотрібно присвоювати їм адреси (клієнтам), як в більшості системах обміну повідомленнями. Для MQTT v3.1.1 існує клієнтське

програмне забезпечення, яке досупне практично на всіх мовах програмування, а також для основних операційних систем Linux, Mac з проекту Eclipse Paho.

Для MQTT v5.0 обмежена підтримка клієнтського ПЗ від Eclipse. На даний момент доступний лише клієнт C. Таблиця порівняння клієнтів і сторінка завантаження знаходяться за адресою: https://www.eclipse.org/paho/-downloads.php.

**MQTT брокери або сервери**

Спочатку використовувався термін “брокер”, але тепер він стандартизований як “сервер”. Існує багато брокерів MQTT, які можна використовувати для тестування і для реальних задач. Існують безкоштовні брокери з самостійним розміщенням, найбільш популярними з них є Mosquitto, а комерційний - HiveMQ.

Mosquitto – це безкоштовний MQTT-брокер з відкритим вихідним кодом, який запускається на операційних системах Windows і Linux. Можна не встановлювати свій власний брокер, а можна використати хмарний брокер від постачальників хмарних послуг, таких як IBM, Microsoft (Azure), ThingsBoard **(**demo.thingsboard.io:1883**)** і т.д. Наприклад Eclipse надає вільно доступний MQTT брокер і сервер COAP, які можна використати для тестування. Адрес iot.eclipse.org і порт 1883 або 8883(SSL).

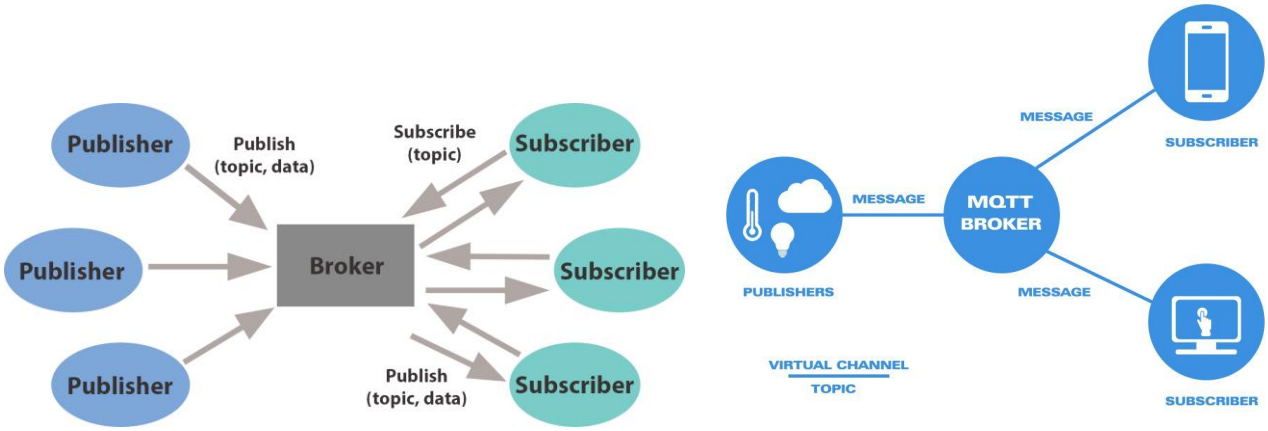



Рис.1.2. MQTT модель “видавець-підписник” (publisher-subscriber)

**MQTT безпека**

MQTT підтримує різні механізми аутентифкації та захисту даних. Ці механізми безпеки конфігуруються на MQTT-брокері, а клієнт повинен дотримуватися встановлених механізмів.

**MQTT через WebSockets**

Websockets дозволяють отримувати MQTT-дані безпосередньо у web-браузері. Це є важливо, оскільки web-браузер може стати інтерфейсом для відображення MQTT-даних. Підтримку MQTT web-сокетів для web-браузерів забезпечує Javascript MQTT клієнт.

Розглянемо детальніше протокол MQTT і як формуються повідомлення або пакети MQTT, а саме:

- формат повідомлення MQTT;
- заголовок повідомлення MQTT;
- поля повідомлень і кодування;
- приклад кодування управляючого повідомлення.

MQTT є бінарним протоколом, в якому управляючі елементи є двійковими байтами, а не текстовими рядками. MQTT використовує команду і формат підтвердження команди. Це означає, що кожна команда має відповідне підтвердження.

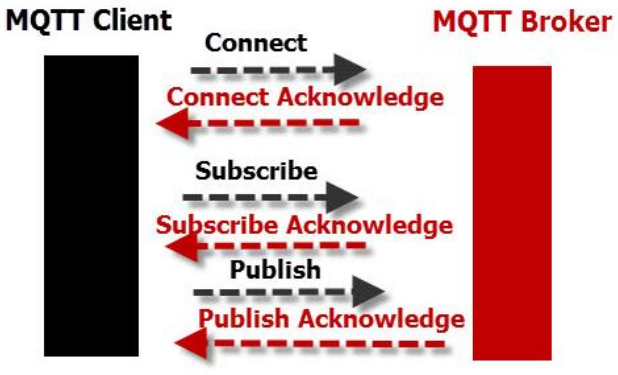



Рис.1.3. Протокол MQTT клієнт-брокер

Назви тем (Topic names), ідентифікатор клієнта (Client ID), імена користувачів (User names) і паролі (Passwords) кодуються у вигляді рядків (стрічок) UTF-8. Корисна нагрузка, виключаючи інформацію протокола MQTT, таку як

ідентифікатор клієнта і т.д., є двійковими даними, а вміст і формат залежать від задачі. Пакет або формат повідомлення MQTT складається з 2 байтового фіксованого заголовку (завжди присутній) + змінного заголовка (незавжди присутній) + корисної нагрузки (незавжди присутня).

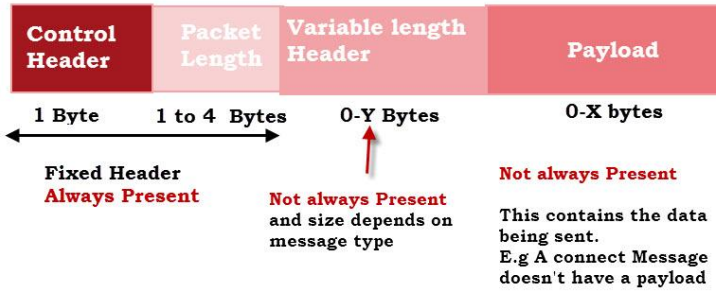



Рис.1.3. Структура стандартного пакету MQTT

Поле фіксованого заголовка складається з поля управління і поля довжини пакета змінної довжини. Мінімальний розмір поля довжини пакета становить 1 байт, який призначений для повідомлень із загальною довжиною менше 127 байтів (не включаючи поля управління і довжину). Максимальний розмір пакета становить 256 МБ. Невеликі пакети менше 127 байтів мають поле довжини пакета в 1 байт.

Пакети більше 127 і менше 16383 використовують 2 байта і т.д. Використовується 7 бітів, а 8-й біт є бітом продовження. Мінімальний розмір пакета становить всього 2 байта з одним полем управління байтами і полем довжини пакета одного байта. Наприклад, повідомлення про роз'єднання займає всього 2 байта.

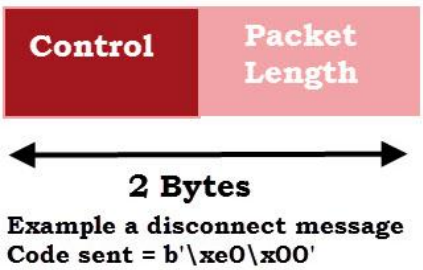



Рис.1.4. Мінімальний пакет MQTT

Восьми бітове поле управління є першим байтом 2-байтового фіксованого заголовка. Він розділений на два 4-бітних поля і містить всі команди і відповіді

протоколу. Перші 4 старших біта є полем типу команди або повідомлення, а решта 4 біти використовуються в якості прапорців управління.

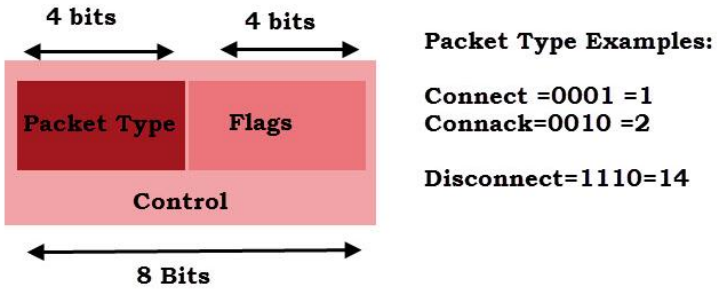



Рис.1.5. Структура MQTT поля управління

В Таблиці нижче взятої з документації на протокол MQTT 3.1.1 показано приклад MQTT команд та їх відповідні коди. Оскільки вони є найбільш значущою частиною 8 бітного байтового поля, тому їх байтові значення зображено в десятковій системі так як вони з'являються в пакеті даних.

**Sample MQTT Control Message**

| Name | Value | Direction of flow | Description | Decimal |
|---|---|---|---|---|
| Reserved | 0 | Forbidden | Reserved | |
| CONNECT | 1 | Client to Server | Client request to connect to Server | 16 |
| CONNACK | 2 | Server to Client | Connect acknowledgment | 32 |
| PUBLISH | 3 | Client to Server or | Publish message | 48 |

=00010000 =16 Dec

=00100000 =32 Dec

**Note: taken from the OASIS MQTT 3.1.1 specification document**

Рис.1.6. Приклад MQTT управляючого повідомлення

**Прапорці управління (Control Flags)**

Хоча є 16 можливих прапорців, лише декілька зазвичай використовуються. Повідомлення, що публікується, найбільше використовує прапорці, які зображено в Таблиці нижче.

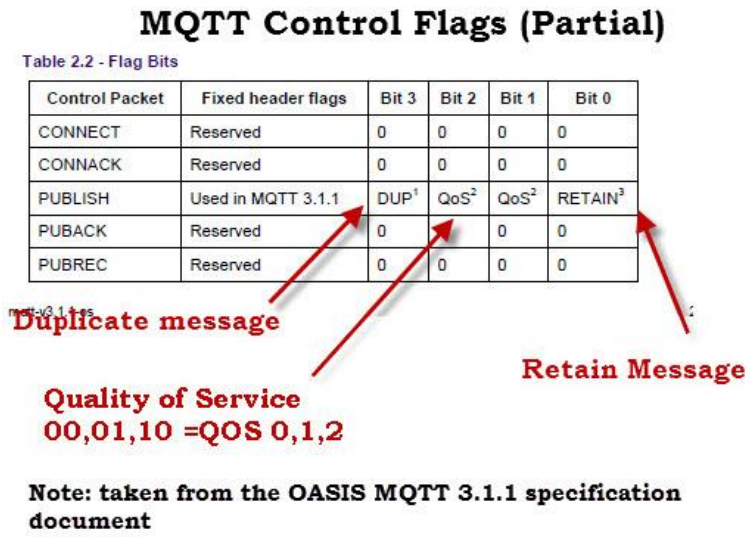



Рис.1.7. MQTT прапорці управління

**Залишкова довжина (Remaining Length)**

Залишкова довжина – це змінна довжина між 1 і 4 байтами. Кожний байт використовує 7 біт довжини з MSB як прапорцем продовження. Залишкова довжина є числом байтів, що включає заголовок змінної довжини і нагрузку (корисну інформацію), як зображено на Рис. 1.8.

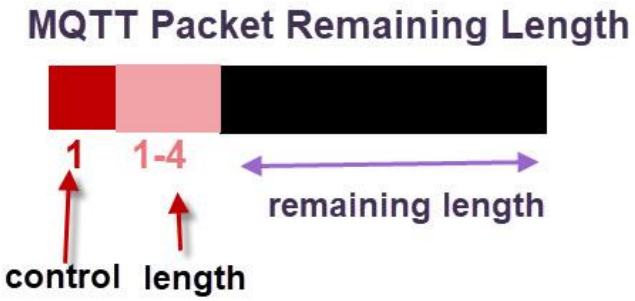



Рис.1.8. Залишкова довжина пакету MQTT

На Рис.1.9 зображено довжину поля для пакету розміром 64 і 321 байти. Залишкова довжина пакету 64 байти з необхідних лише 1 байт.

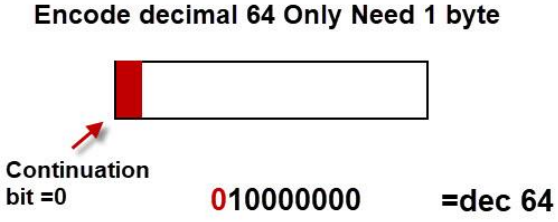



Рис.1.9. Приклад декодування поля залишкової довжини

Довжина пакету 321 байт потребує 2 байтове поле залишкової довжини:

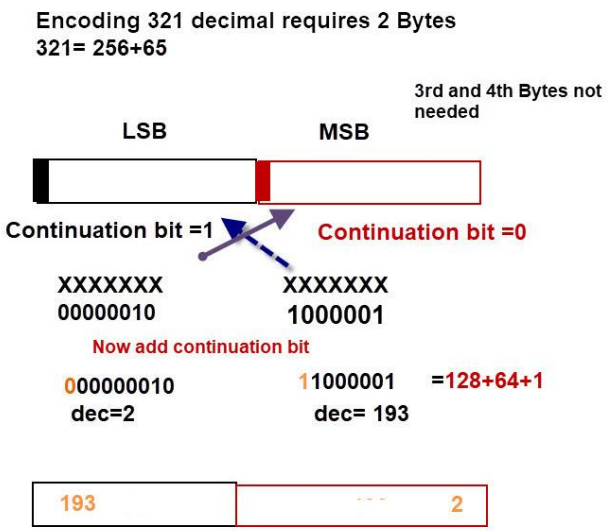



Рис.1.10. Приклад декодування поля залишкової довжини

На Рис.1.11 показано розміри пакету і довжину поля пакету:

**Remaining Length Field Values**

| | | |
|---|---|---|
| 1 | 0 (0x00) | 127 (0x7F) |
| 2 | 128 (0x80, 0x01) | 16 383 (0xFF, 0x7F) |
| 3 | 16 384 (0x80, 0x80, 0x01) | 2 097 151 (0xFF, 0xFF, 0x7F) |
| 4 | 2 097 152 (0x80, 0x80, 0x80, 0x01) | 268 435 455 (0xFF, 0xFF, 0xFF, 0x7F) |

Рис.1.11. Значення поля залишкової довжини

**Приклад MQTT повідомлення підключення і відключення**

В якості ілюстрації розглянемо деталі пакету для повідомлення підключення. На Рис.1.12 показано типи повідомлень управління та їх шістнадцяткові коди.

**MQTT Message Types and Hex Codes**

```
# Message types
CONNECT = 0x10          =16 decimal
CONNACK = 0x20
PUBLISH = 0x30
PUBACK = 0x40
PUBREC = 0x50
PUBREL = 0x60
PUBCOMP = 0x70
SUBSCRIBE = 0x80      =128 decimal
SUBACK = 0x90
UNSUBSCRIBE = 0xA0
UNSUBACK = 0xB0
PINGREQ = 0xC0
PINGRESP = 0xD0
DISCONNECT = 0xE0       =224 decimal
```

Рис.1.12. Типи MQTT-повідомлень та їх шістнадцяткові значення

Нижче наведено приклад реального підключення та відключення клієнта, що показує фактичні значення байтів для відправлених і отриманих даних.

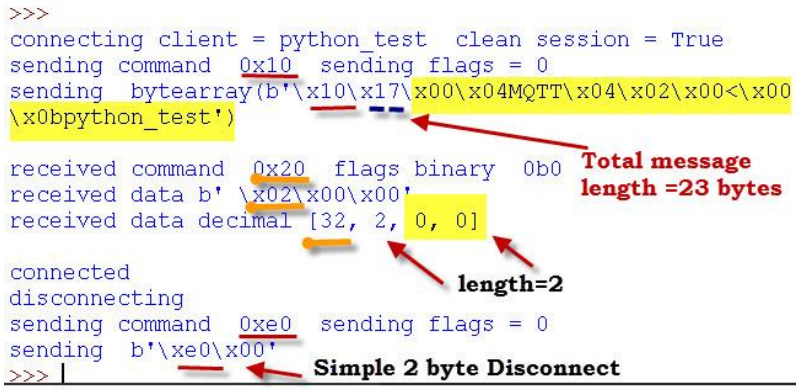



Рис.1.13. Приклад MQTT-повідомлення

**Змінна довжина заголовка**

Поле змінної довжини заголовку незавжди присутнє в повідомленні MQTT. Відомі типи MQTT повідомлень і команд потребують використання цього поля, що нести додаткову управляючу інформацію. MQTT пакет = управління + довжина + назва протоколу + рівень протоколу + прапорці зв'язку (підключення) + нагрузка. Цікаво те, що поле client ID відсилається як перша частина нагрузки, і не є частиною заголовка. Воно має поле довжиною 2 байти як префікс. Payload=client ID = довжина client ID + client ID.

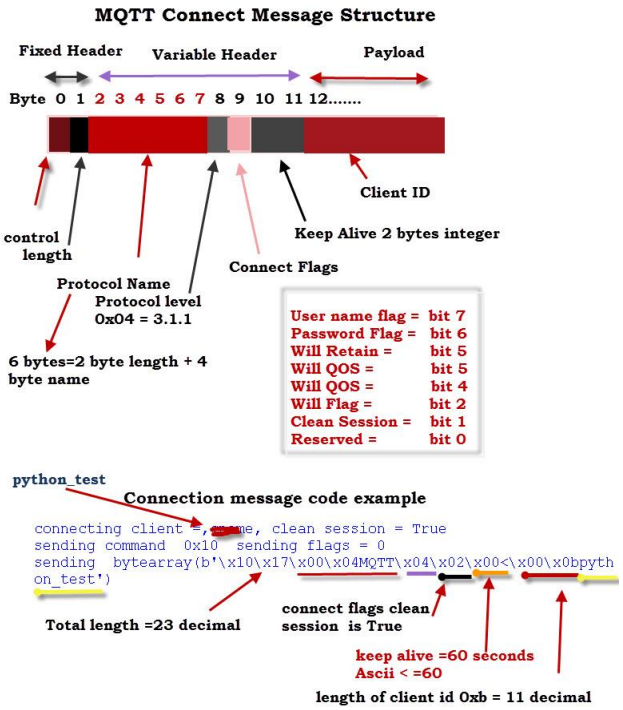



Рис.1.14. Структура MQTT-пакету підключення

На Рис.1.15 зображено передачу MQTT пакетів даних відловлених за допомогою програми-аналізатора трафіку Wireshark Network Protocol Analyzer.

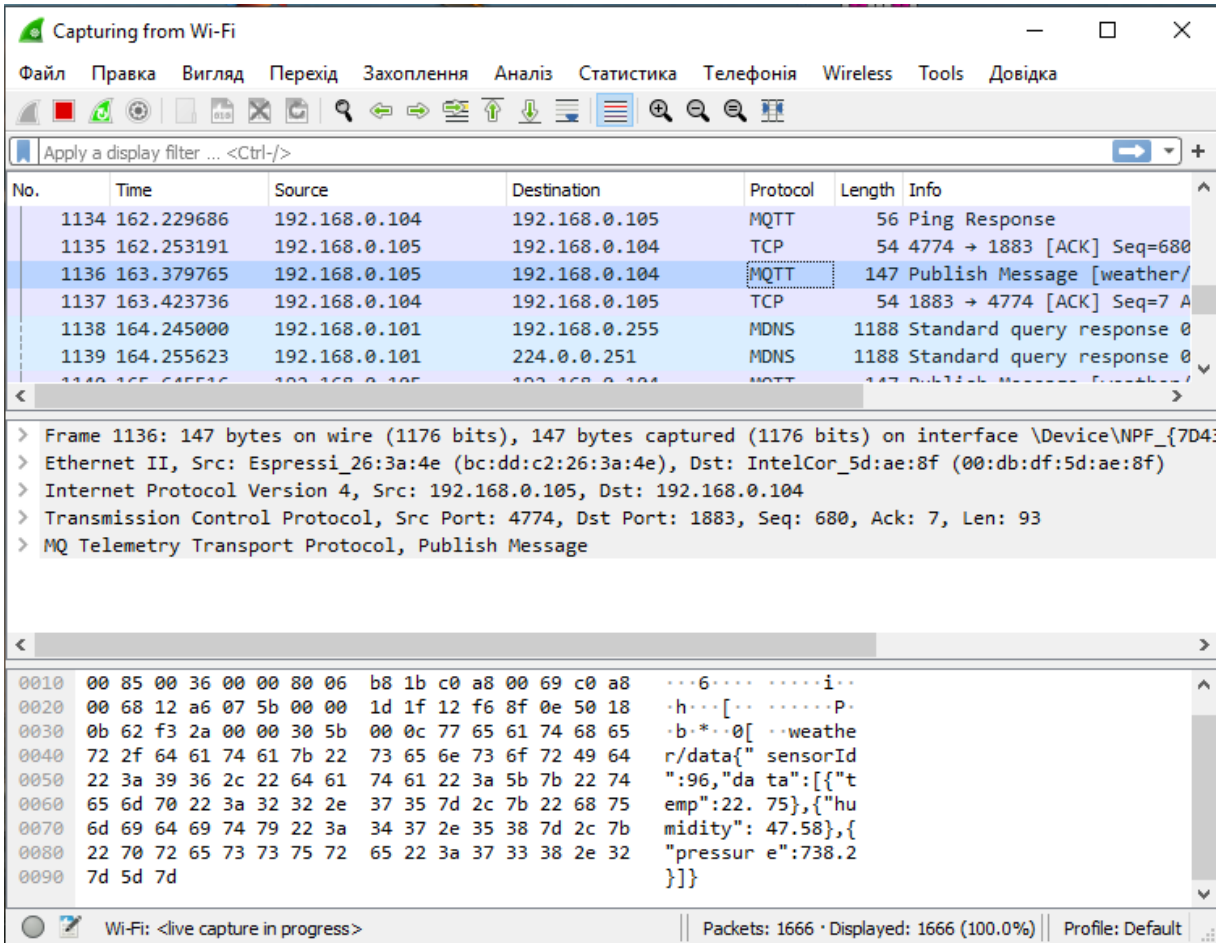



Рис.1.15. Wireshark Packet Capture (Wireshark Network Protocol Analyzer) – MQTT

Wireshark Network Protocol Analyzer показує підключення і публікації MQTT-клієнта. Можна побачити ACK пакети, які мають загальну довжину пакету 54 байта. Слід також відмітити, що кожна MQTT команда або відповідь отримує TCP ACK і можливо також навіть MQTT ACK. На Рис.1.15 видно передачу MQTT – пакету даних розміром 147 байтів від клієнта (4774) на сервер (1883.

MQTT топіки (теми) є формою адресації, що дає змогу MQTT клієнтам ділитися (обмінюватися) інформацією. Теми (топіки) MQTT структуровані в ієрархії, аналогічно каталогам і файлам у файловій системі, використовують косу лінію (/) в якості розділювача. Використовуючи цю систему, можна створювати зручні і зрозумілі структури іменовані за своїм вибором. Наприклад, назви топіків (тем):

- враховують регістр (мала чи велика буква);
- використовують стрічки (рядки) у форматі UTF-8;
- мають складатися щонайменше з одного символу, щоб бути допустимими.

За виключенням топіка (теми) $SYS, нема стандартної або за замовчуванням структури топіка (теми). Тобто нема ніяких топіків (тем) створених в брокері за замовчуванням, за виключенням теми $SYS. Всі топіки (теми) створюються клієнтом-підписником або клієнтом-видавцем і не є постійними. Топік (тема) існує лише в тому випадку, якщо клієнт підписався на нього (неї) або брокер має збережене або останні бажанні повідомлення збережені для цього топіка (теми).

Топік (тема) $SYS. Це зарезервований топік, який використовуються більшістю брокерів MQTT для публікування інформації про брокер. Ці топіки MQTT - клієнти можуть лише читати. Нема стандарту структури цього топіка, але є вказівки (рекомендації), яким, як правило, слідує більшість реалізацій брокера. Наприклад, панель Mosquitto, яка відображає дані опубліковані на топік $SYS брокером Mosquitto.

**Підписка на топіки (теми)**

Клієнт може підписатися на індивідуальну або множинну тем (топіків). При підписці на множину топіків, використовується символи підстановки. Вони є наступними:

- **# (хеш символ)** – багаторівневий символ підстановки;
- **+ (символ плюса)** –однорівневий символ підстановки.

Символи підстановки можуть бути лише використані для позначення рівня або множини рівнів, наприклад /house/# і не є як частина імені для позначення множини символів наприклад запис hou# не є коректним.

Назви топіків (тем). Приклади:

**Дійсні описи (найменування) топіків (тем):**

Підписки на один топік (тему):

- /
- /house
- house/room/main-light

- house/room/side-light

**Використання символу підстановки**

Підписка на тему (топік) house/# покриває: house/room1/main-light; house/room1/alarm; house/garage/main-light; house/main-door.

Підписка на тему (топік) покриває house/+/main-light; house/room1/main-light; house/room2/main-light; house/garage/main-light, але не покриває house/room1/side-light; house/room2/side-light.

**Недійсні (некоректні) записи на підписку топіків:**

- house+ – причина- не вказано рівень топіку;
- house# – причина – не вказано рівень топіку.

**Публікування на теми (топіки)**

Клієнт може публікувати лише на індивідуалльну тему (топік). Тому, використовуючи спеціальні символи коли публікування не дозволено. Наприклад, щоб опублікувати повідомлення на два топіки, потрібно опублікувати повідомлення двічі.

**Створення топіків (тем):**

Топіки (теми) створюються динамічно: 1) якщо хтось підписується на топік; 2) якщо хтось публікую повідомлення на топік зі збереженим повідомленням встановленим в true.

**Видалення топіків з брокера:** 1) якщо останній клієнт, що підписується до цього брокера, відключається, і очистка сесії є true; 2) якщо клієнт підключається з очищенням сесії і встановленням в true.

**Перевидання (повторне опублікування) даних топіка**

Це може бути зроблено при зміні або комбінуванні схем найменування. Ідея полягає в тому, що клієнт підписаний на топік, наприклад hub1/sensor1 і перевидає (публікуає заново) дані використовуючи новий топік, який називається наприклад house1/main-light.

**Quality of Service (QoS)**

Quality of Service (QoS) (якість обслуговування) рівень є угодою між відправником повідомлення і одержувачем повідомлення, що визначає гарантію доставки відповідного повідомлення. В MQTT є три рівні QoS:

- максимум один раз - At most once (0);
- хоча би один раз - At least once (1);
- точно один раз - Exactly once (2).

Коли говоримо про QoS в MQTT, то потрібно розглядати дві сторони доставки повідомлення:

1. Доставка повідомлення від клієнта-відправника (клієнта-видавця) до брокера.
2. Доставка повідомлення від брокера до клієнта-підписника (клієнта-одержувача).

# РОЗДІЛ ІІ. ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ IoT - СИСТЕМИ МОНІТОРИНГУ МЕТЕОПАРАМЕТРІВ

## 2.1. Система автоматизованого проектування та моделювання мікроконтролерних пристроїв Proteus

Proteus VSM – це потужна система автоматизованого проектування, що дозволяє провести віртуальне моделювання роботи різних аналогових та цифрових пристроїв. В програмному пакеті Proteus VSM можна зібрати схему будь-якого електронного пристрою та провести моделювання його роботи. Помилки можуть бути виявлені на стадії проектування і трасування. Програма складається з двох модулів. ISIS – редактор електричних схем з подальшою імітацією їх роботи. ARES – редактор друкованих плат, що включає автотрасувальник Electra, вбудований редактор бібліотек і автоматичну систему розміщення компонентів на платі. Окрім цього, ARES може створити тривимірну модель друкованої плати.

Proteus VSM включає в собі понад 6000 електронних компонентів з даними для довідки, а також демонстраційні ознайомчі проекти. Програма має інструменти USBCONN і COMPIM, які дозволяють підключити віртуальний пристрій до портів USB і COM комп’ютера. При підключенні до цих портів будь-якого зовнішнього приладу віртуальна схема буде працювати з ними, так ніби вона існує в реальності. Proteus VSM підтримує наступні компілятори: CodeVisionAVR і WinAVR (AVR), ICC (AVR, ARM7, Motorola), HiTECH (8051, PIC Microchip) і Keil (8051, ARM). Існує можливість експорту моделей електронних компонентів з програми PSpice.

Не дивлячись на те, що програма працює з пристроями, які складаються з декількох мікроконтролерів і навіть з чіпами від різних виробників в одному пристрої. Слід розуміти, що симуляція повторює роботу реальної схеми не абсолютно точно. Proteus VSM є комерційним програмним продуктом. З сайту розробника компанії Labcenter Electronics https://www.labcenter.com/simulation/ (https://www.labcenter.com/downloads/) можна завантажити безкоштовну демонстраційну версію.

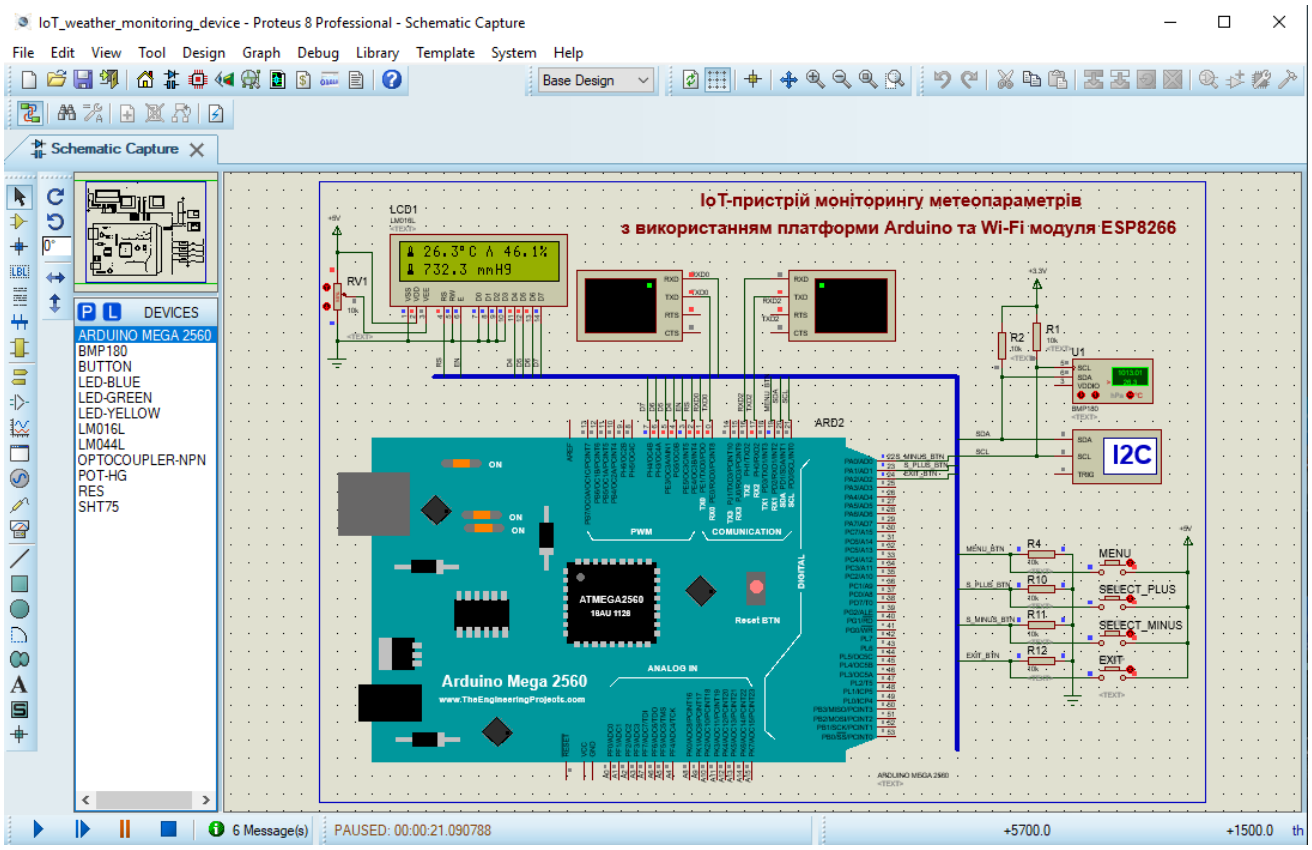



Рис.2.1. Середовище Proteus ISIS

## 2.2. Середовище розробки програм для МК платформи Arduino – Arduino IDE

Розробку програм для МК плати Arduino проводять в безкоштовному програмному середовищі Arduino IDE. Середовище Arduino IDE дозволяє написати програму, зкомпілювати її та завантажити прошивку у флеш-пам’ять мікроконтролера плати Arduino. В Arduino IDE використовується мова Processing/Wiring, яка є спрощеним варіантом мови C++ (її діалект). Останню версію Arduino IDE для операційних систем Windows, Mac OS X, Linux та Linux ARM можна завантажити з сторінки: https://www.arduino.cc/en/main/software.

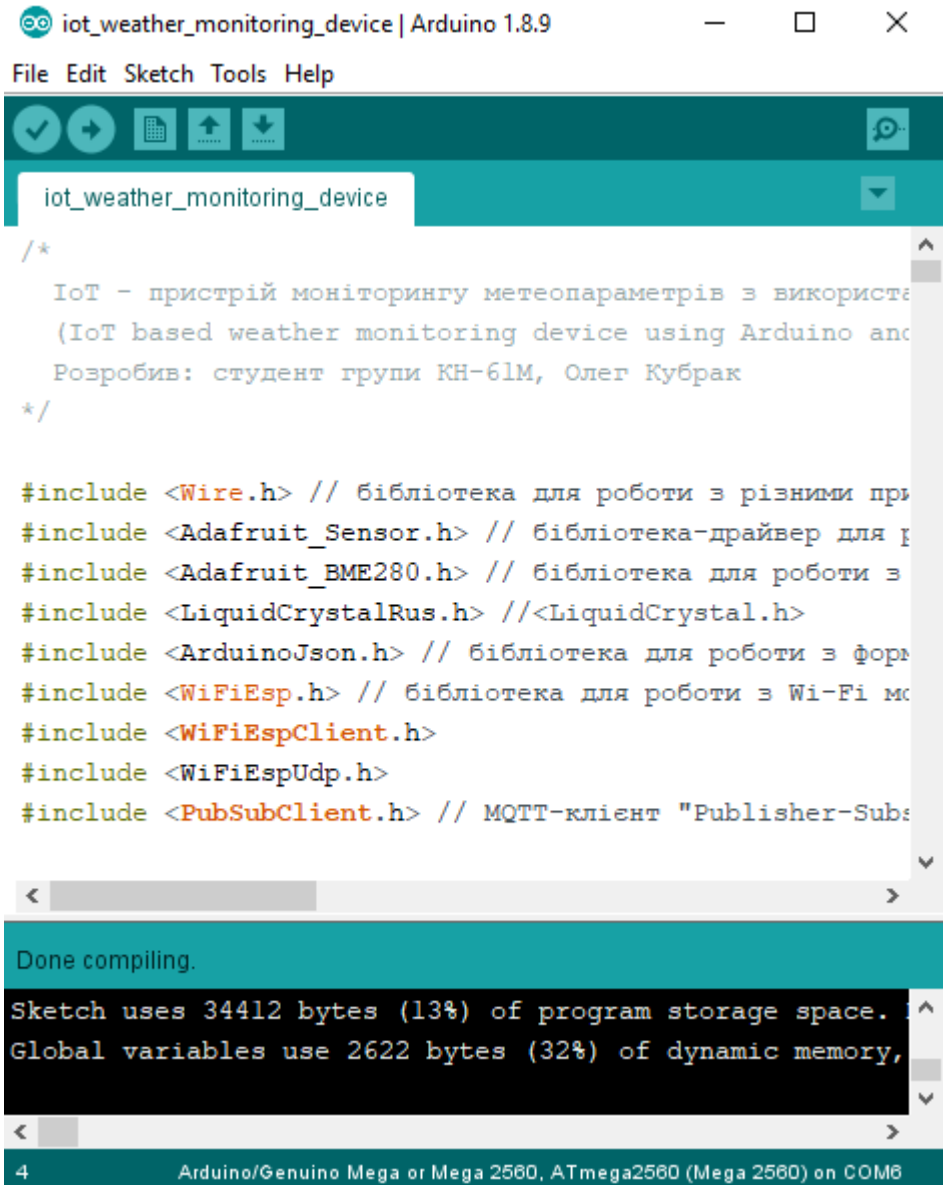



Рис.2.2. Середовище розробки ПЗ під платформу Arduino - Arduino IDE

**Налаштування середовища Arduino IDE**

Arduino IDE складається з редактора програмного коду, області повідомлень, вікна текстового вводу, інструментальної панелі з кнопками для виклику найчастіше використовуваних команд, а також з декількох додаткових меню. В Arduino IDE програма називається скетчом (sketch). Скетч пишуть в текстовому редакторі з колірною підсвіткою коду програми. У області повідомлень при зберіганні та експорті проекту виводиться інформація про помилки та пояснення. У вікні текстового виводу виконується вивід Arduino - повідомлень з детальними звітами про помилки та іншою інформацією. Кнопки на панелі

інструментів призначені для перевірки та запису програми, створення, відкриття і зберігання скетчу, відкриття Serial monitor (монітору послідовного інтерфейсу).

У розроблювальній програмі (sketch) можна додати необхідний функціонал включивши відповідні бібліотеки. На сьогодні є багато різних програмних бібліотек, що дають змогу підключити до Arduino різні давачі, шилди, дисплеї, інтерфейси тощо. Інтегроване середовище розробки Arduino IDE включає в себе набір стандартних бібліотек Serial, Wire, SPI, EEPROM та інші. Бібліотеки знаходяться в підкаталозі бібліотек libraries Arduino - каталогу. Потрібна бібліотека завантажується з відповідного інтернет ресурсу, а потім копіюється в каталог зі стандартними бібліотеками. В бібліотечному каталозі знаходиться заголовний файл з розширенням *.h та файл з вихідним кодом бібліотеки з розширенням *.cpp або *.c. В бібліотеках зазвичай є приклади їх використання. Вони знаходяться у підкаталозі examples бібліотечного каталогу. Якщо інсталяція бібліотеки пройшла успішно, то вона має з’явитися у меню Sketch|Include Library. Бібліотеку включають у програмний код (sketch) за допомогою запису директиви #include з назвою заголовного файлу бібліотеки. Наприклад: #include <Adafruit_BME280.h>, #include <LiquidCrystal.h>. Заголовний файл бібліотеки містить опис констант, функцій, об’єктів, які можна використовувати у програмному коді (скетчі). Arduino IDE компілює скетч з включеною бібліотекою. Скетч можна завантажити у МК плати Arduino вибравши у меню Засоби|Плата (Tools|Board) модель Arduino - плати та порт підключення Засоби|Порт (Tools|Port). В процесі завантаження програми у МК плати Arduino мигають світлодіоди RX і TX. Скетч прошивається за допомогою спеціального завантажувача (bootloader). Bootloader є невеликою програмою, яка зашита у флеш-пам’ять МК. Вона запускається відразу після включення живлення плати Arduino, або коли на платі натиснута кнопка Reset, або коли комп’ютер-хост USB, до якого підключена плата Arduino видав сигнал скидання (це виконується за допомогою спеціальної схеми реалізації віртуального USB COM-порта Arduino). У програми завантажувача є лише одна основна функція – прийняти через USART від комп’ютера-хоста нову програму, і прошити її у

флеш-пам’ять МК плати Arduino. Монітор послідовного інтерфейсу (монітор порта) (Serial Monitor) дозволяє вивести значення сигналів та дані з різних давачів і модулів підключених до плати Arduino. Монітор послідовного порта дає змогу прийняти та відправити різні дані на цей самий порт. Щоб відіслати дані потрібно ввести у відповідне поле текст і натиснути кнопку Send (відіслати) або клавішу <Enter>. Для коректного використання монітора послідовного порта, потрібно, щоб швидкість передачі даних прописана в скетчі співпадала зі швидкістю встановленою в самому моніторі послідовного порта.

**Структура Arduino-програми (Arduino-скетчу)**

Програмування плат Arduino базується на використанні мов C/C++. Використовується спеціалізований для AVR - мікроконтролерів варіант C/C++ та компілятор AVR - gcc.

Arduino - програма (sketch) складається з двох обов’язкових частин: функцій setup() і loop(). До визначення функції setup() оголошують змінні та підключають потрібні бібліотеки. У функції setup() проводять початкову ініціалізацію змінних, налаштування режимів роботи портів та інші підготовчі дій перед виконанням функції основного циклу програми loop(). Вона завжди є в програмі, навіть якщо не виконує жодних дій. Функція loop() реалізує нескінченний цикл програми. В цьому циклі послідовно виконуються команди описані в тілі функції.

# РОЗДІЛ ІІІ. АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ IoT - ПРИСТРОЮ МОНІТОРИНГУ МЕТЕОПАРАМЕТРІВ

## 3.1 Вибір електронних комплектуючих для проектування IoT – пристрою моніторингу метеопараметрів

Для проектування апаратної частини IoT – пристрою моніторингу метеопарамерів було вибрано наступні електронні компоненти:

- ✓ плата Arduino Mega2560 на 8-розрядному МК AVR ATmega2560;
- ✓ модуль символьного РК-дисплею 16×2 на базі контролера Hitachi HD44780;
- ✓ модуль з цифровим сенсором тиску, температури і вологості BME280;
- ✓ Wi-Fi модуль ESP-01 на базі мікросхеми ESP8266;
- ✓ модуль живлення 5 В/3,3 В;
- ✓ 4 двохконтактні кнопки на 4 виводи.

### Огляд платформи Arduino

Arduino – це апаратно-програмна платформа для розробки різних автоматизованих та робототехнічних систем. Програмна частина складається з безкоштовного програмного середовища Arduino IDE, в якому можна писати, компілювати програму та прошивати її в МК плати Arduino. Апаратна частина складається з набору друкованих плат з мікроконтролерами. Плати виготовляють та продають як офіційні так і сторонні виробники. Arduino побудована на відкритій архітектурі, що дозволяє вільно копіювати або доповнювати лінійку її продукції.

На сьогодні є наступні моделі плат Arduino: Arduino Due на основі 32 - розрядного МК Atmel AT91SAM3X8E ARM Cortex-M3; Arduino Leonardo на базі 8-розрядного МК ATmega32u4; Arduino Uno R3 на базі 8 - розрядного МК AVR ATmega328; Arduino Duemilanove – це пристрій на базі МК ATmega168 або ATmega328; Arduino Micro – це пристрій на основі МК ATmega32u4, розроблений спільно з Adafruit. Arduino Micro використовує МК ATmega32u4 з вбудованим контролером USB. Таке рішення виключає необхідність

використання додаткового контролера, і при підключенні до ПК дозволяє Arduino Micro розпізнаватися в системі як звичайна мишка, клавіатура або віртуальний COM-порт; Arduino Nano – це повноцінний мініатюрний пристрій на базі МК ATmega328 (Arduino Nano 3.0) або ATmega168 (Arduino Nano 2.x) адаптований для використання з макетними платами. Він підключається до ПК по кабелю USB Mini-B; Arduino Mega ADK – це пристрій на основі МК AVR ATmega2560 з використанням мікросхеми MAX3421E, в якій реалізовано USB-хост для підключення смартфонів на базі операційної системи Android; Arduino Mega2560 R3 на базі МК AVR ATmega2560 з використанням МК ATmega16U2 (ATmega8U2 у версях плат R1 і R2) для послідовного з'єднання по USB-порту, Mega на МК ATmega1280; Arduino Fio – це пристрій на базі МК ATmega328P, що працює на частоті 8 МГц від 3,3В. Arduino Fio орієнтований на використання в задачах, що використовують безпровідний зв'язок; Arduino Ethernet – це пристрій на основі МК ATmega328. Він має 14 цифрових виводів, 6 аналогових входів, кварцовий резонатор на 16 МГц, роз'єм RJ45, роз'єм живлення, роз'єм для внутрішньо схемного програмування ICSP, а також кнопка скидання; Arduino LilyPad – портативна, мобільна плата на МК ATmega168V або ATmega328V, яка розроблена спеціально для використання з предметами одягу і текстилю; Arduino Fio - плата на МК ATmega328P, яка орієнтована на використання у задачах, що використовують безпровідний зв'язок; Arduino Mini – це найменша плата з лінійки плат Arduino, яка орієнтована на викортсання з макетними платами або в задачах з високими вимогами до габаратних розмірів; Arduino Pro – плата на базі МК ATmega168 або ATmega328, яка призначена для використання у великих професійних проектах; Arduino Pro Mini – це плата на базі МК ATmega328 як і плата Pro призначена для напівстаціонарного монтажу в різне обладнання або установки. По розміщенню виводів Arduino Mini Pro сумісна з Arduino Mini.

**Arduino Mega2560**

На платі Arduino Mega2560 (Рис.3.1) передбачено все необхідне для зручної роботи з мікроконтролером: 54 цифрових входа/вихода, 16 аналогових входів,

роз’єм для програмування USB, зовнішній роз’єм живлення і кнопка скидання (RESET).

**Характеристики плати Arduino Mega2560:**

- Мікроконтролер: ATmega2560
- Ядро: 8-бітний AVR мікроконтролер
- Тактова частота: 16 МГц
- Об’єм Flash-пам’яті: 256 КБ (8 КБ займає завантажувач (bootloader))
- Об’єм SRAM-пам’яті: 32 КБ
- Об’єм EEPROM-пам’яті: 4 КБ
- Портів вводу-виводу всього: 54
- Портів з АЦП: 16
- Розрядність АЦП: 10 біт
- Портів з ШІМ: 15
- Розрядність ШІМ: 8 біт
- Апаратних інтерфейсів SPI: 1
- Апаратних інтерфейсів $I^2C$/TWI: 1
- Апаратних інтерфейсів UART/Serial: 4
- Номінальна робоча напруга: 5 В
- Максимальний вихідний струм піна 5V: 800 мА
- Максимальний вихідний струм піна 3V3: 150 мА
- Максимальний струм з піна або на пін: 40 мА
- Допустима вхідна напруга від зовнішнього джерела живлення: 7-12 В
- Габарити: 101×53 мм.

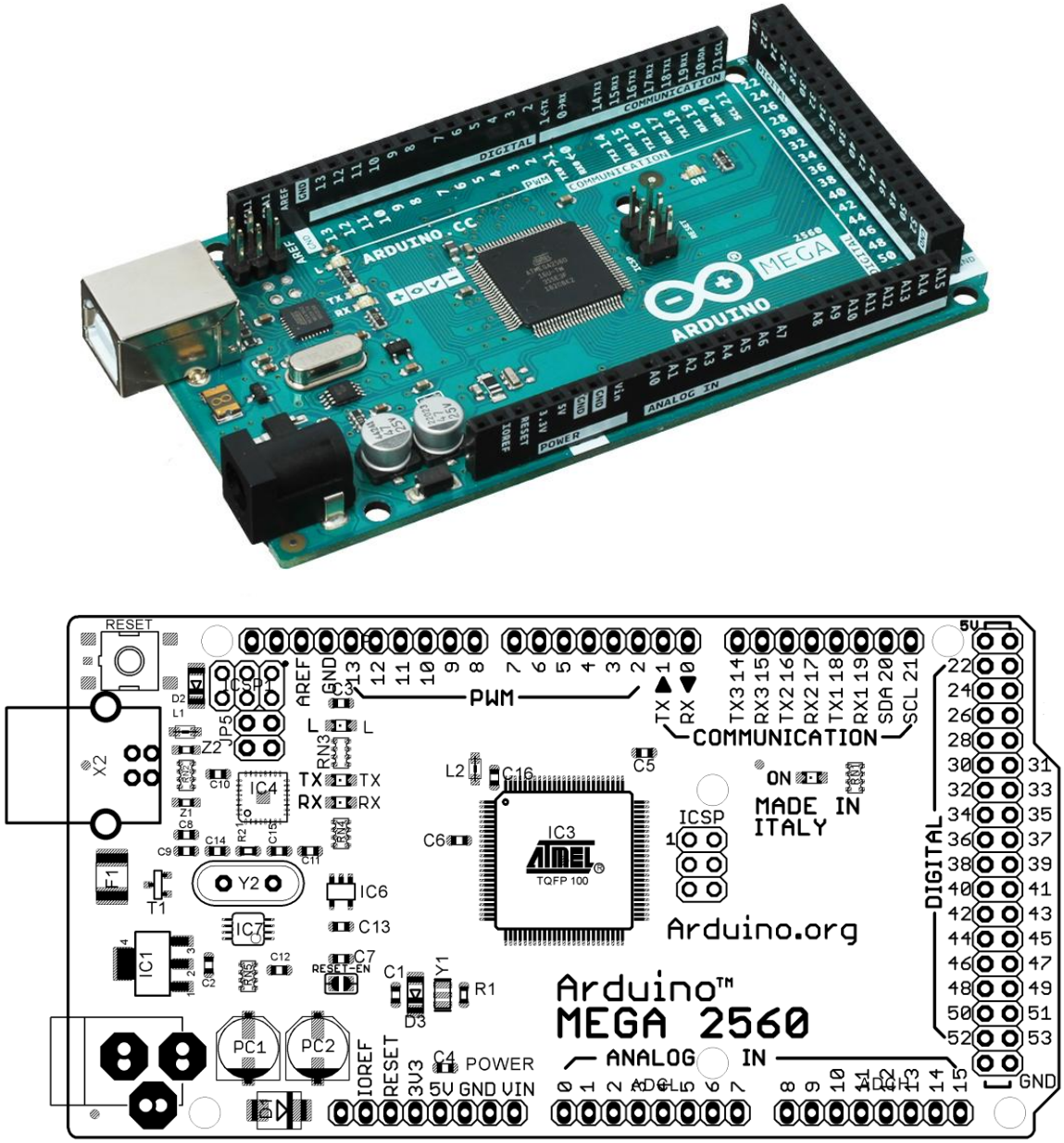



Рис.3.1. Плата Arduino Mega2560

На платі Arduino Mega2560 є наступні елементи:

- **8-бітний AVR мікроконтролер ATMega2560.** Він є серцем платформи Arduino Mega. Контролер з тактовою частотою 16 МГц має 256 КБ Flash-пам’яті для зберігання прошивки, 8 КБ оперативної пам’яті SRAM і 4 КБ енергонезалежної пам’яті EEPROM для зберігання даних.
- **8-бітний AVR мікроконтролер ATMega16U2.** Мікроконтролер ATmega16U2 забезпечує зв’язок мікроконтролера ATMega2560 з USB – портом комп’ютера.

При підключенні до ПК Arduino Mega2560 розпізнається (визначається) як віртуальний COM-порт.

Табл.3.1. Світлодіодна індикація на платі Arduino Mega2560

| Назва світлодіода | Призначення |
|---|---|
| RX і TX | Блимають при обміні даними між Arduino Mega2560 і ПК. |
| L | Користувацький світлодіод підключений до 13 піна мікроконтролера плати Arduino. При високому рівні світлодіод засвічується (включається), а при низькому – гасне (виключається). |
| ON | Присутнє живлення на платі Arduino Mega. |

- **Роз'єм USB.** Роз'єм USB Type-B для живлення прошивки платформи Arduino Mega2560 за допомогою ПК.
- **Роз'єм зовнішнього живлення.** Роз'єм для підключення зовнішнього живлення від 7 В до 12 В.
- **Кнопка скидання (RESET).** Аналог кнопки RESET звичайного ПК. Призначена для скидання (перезавантаження) мікроконтролера.
- **Регулятор напруги 5 В.** Лінійний понижуючий регулятор напруги LD1117S50CTR з виходом 5 В забезпечує живленням мікроконтролери ATmega2560, ATmega16U2 та іншу логіку платформи. Максимальний вихідний струм становить 800 мА.
- **Регулятор напруги 3,3 В.** Лінійний понижуючий регулятор напруги LP2985-33DBVR з виходом 3,3 В. Лінія виведена тільки на пін 3V3. Максимальний вихідний струм становить 150 мА.
- **Роз'єм ICSP.** Роз'єм ICSP призначений для внутрішньо схемного програмування (всередині схеми) мікроконтролера ATmega2560. Також з використанням бібліотеки SPI дані виводи можуть використовуватися для підключення з платами розширення по інтерфейсу SPI. Лінії SPI виведені на 6-контактний роз'єм, а також продубльовані на цифрових пінах 50 (MISO), 51 (MOSI), 52 (SCK) і 53 (SS).

- **Роз’єм ICSP1.** Роз’єм ICSP1 для внутрішньо схемного програмування мікроконтролера ATmega16U2.

## Розпіновка плати Arduino Mega2560

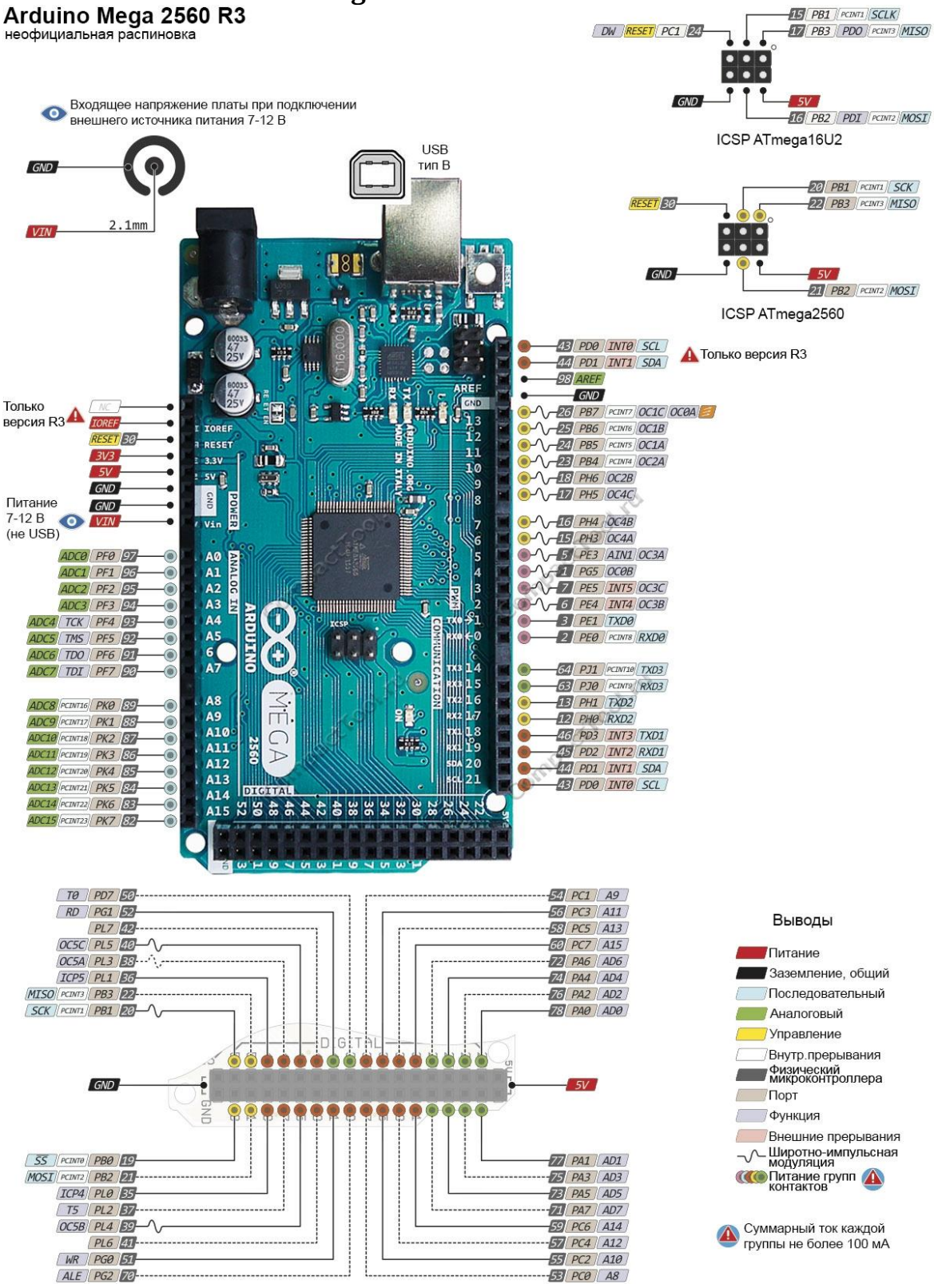



Рис.3.2. Розпіновка плати Arduino Mega2560

**Піни живлення:**

- **VIN:** Вихідний пін для підключення зовнішнього джерела живлення з напругою в діапазоні від 7 до 12 В. Через контакт можна використовувати (брати) напругу, коли пристрій заживлено через зовнішній роз'єм живлення.
- **5V:** Вихідний пін від регулятора напруги на платі з виходом 5 В і максимальним струмом 800 мА. Живити пристрій через вивід 5V не рекомендовано – можна спалити плату.
- **3.3V:** Вихідний пін від регулятора напруги з виходом 3,3 В і максимальним струмом 150 мА. Брати живлення з виводу 3V3 не рекомендовано – можна спалити плату.
- **GND:** Виводи землі.
- **IOREF:** Контакт надає платам розширення інформацію про робочу напругу мікроконтролера. В залежності від напруги, плата розширення може переключитися на відповідне джерело живлення або задіяти перетворювачі рівнів.
- **AREF:** Пін для підключення зовнішньої опорної напруги АЦП відносно якої відбуваються аналогові вимірювання при використанні функції analogReference() з параметром "EXTERNAL".

**Порти вводу/виводу**

- **Цифрові входи/виходи:** піни 0-53. Логічний рівень одиниці – 5 В, нуля – 0 В. Максимальний струм виходу – 40 мА. До контактів підключені підтягуючі резистори, які за замочуванням виключені, але можуть бути включені програмно.
- **ШІМ (PWM):** піни 2-13 і 44-46. Дозволяють виводити аналогові значення у вигляді ШІМ-сигналу. Розрядність ШІМ не змінюється і встановлена 8 біт.
- **АЦП:** піни A0-A16. Дозволяє представити аналогову напругу в цифровому вигляді. Розрядність АЦП не міняється і встановлена в 10 біт. Діапазон вхідної

напруги від 0 до 5 В. При подачі великої напруги можна спалити мікроконтролер.

- **TWI/I$^2$C:** піни 20 (SDA) і 21 (SCL). Для обміну даними з периферією по інтерфейсу I$^2$C. Для роботи використовується Arduino-бібліотека Wire.
- **SPI:** піни 50 (MISO), 51 (MOSI), 52 (SCK) і 53 (SS). Для комунікації з периферією по інтерфейсу SPI. Для роботи використовується бібліотека SPI.
- **UART:** піни 0 (RX) і 1 (TX), 19 (RX1) 18 (TX1), 17 (RX2) і 16 (TX2), 15 (RX3) і 14 (TX3). Використовується для комунікації плати Arduino з ПК або іншими пристроями по послідовному інтерфейсу. Виводи 0 (RX) і 1 (TX) з'єднанні з відповідними виводами мікроконтролера ATmega16U2, які виконують роль USB-UART перетворювача. Для роботи з послідовним інтерфейсом використовується бібліотека Serial.

**Опис цифрового сенсора тиску, температури і вологості BME280**

Сенсор BME280 призначений для вимірювання атмосферного тиску, температури і вологості. Він є черговим сенсором тиску фірми Bosch Sensortec. який у порівнянні з першими сенсорами серії (BMP085 і BMP180) має кращі характеристики та менші розміри. Його відмініть від сенсора BMP280 це наявність гігрометра, що дозволяє вимірювати відносну вологість повітря і створити на ньому маленьку метеостанцію. На Рис.3.4 зображено модуль з цифровим сенсором тиску, температури і вологості BME280.

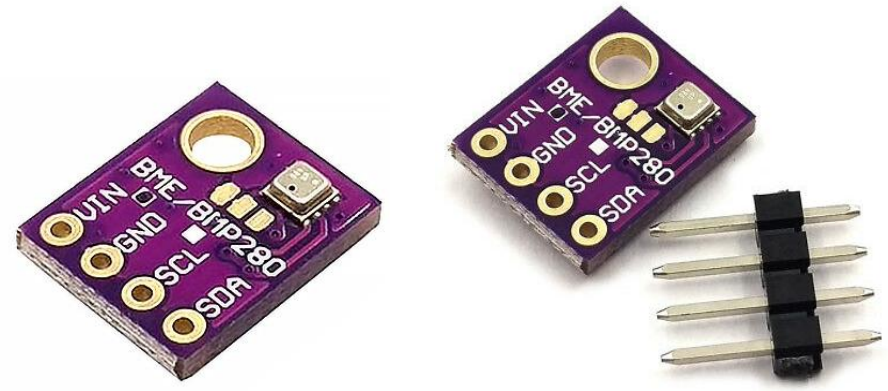



Рис.3.4. Модуль з цифровим сенсором атмосферного тиску, температури і вологості BME280

**Технічні характеристики сенсора BME280**

- Інтерфейс підключення: SPI, $I^2C$;
- Напруга живлення: 3,3 В;
- Діапазон вимірювання тиску: 300 – 1100 гПа;
- Діапазон вимірювання температури: -40 - +85 °С;
- Діапазон вимірювання вологості: 0 - 100 %;
- Енергоспоживання: режим вимірювань – 2,74 нА; в режимі сну: - 0,1 нА
- Точність вимірювань: тиск – 0,01 гПа ( < 10 см), температура – 0,01 °С, вологість – 3 %.

Модуль побудований на цифровому сенсорі температури, вологості і тиску нового покоління BME280, який виготовляє фірма Bosch Sensortec. Він є наступником таких давачів як BMP180, BMP085 і BMP183.

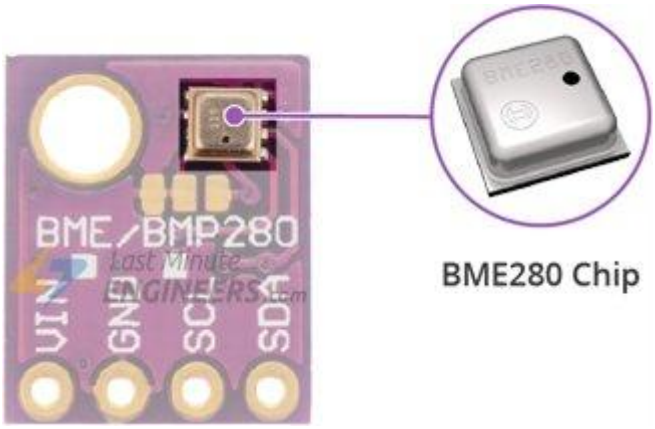



Рис.3.5. Чіп BME280

Цей точний сенсор може вимірювати відносну вологість від 0 до 100% з точністю ±3%, атмосферний тиск від 300 Па до 1100 гПа з абсолютною точністю ±1 гПа, і температуру від -40 °С до 85 °С з точністю ±1,0 °С.

Вимірювання тиску є настільки точними (низький висотний шум 0,25 м), що його можна використовувати як висотомір з точністю ±1 м.

**Вимоги до живлення**

Модуль має на платі 3,3 В регулятор напруги LM6206 і $I^2C$ перетворювач (конвертер) рівнів сигналів, тому його можна безпосередньо підключати до мікроконтолерів з 3,3 В або 5 В логікою, наприклад Arduino.

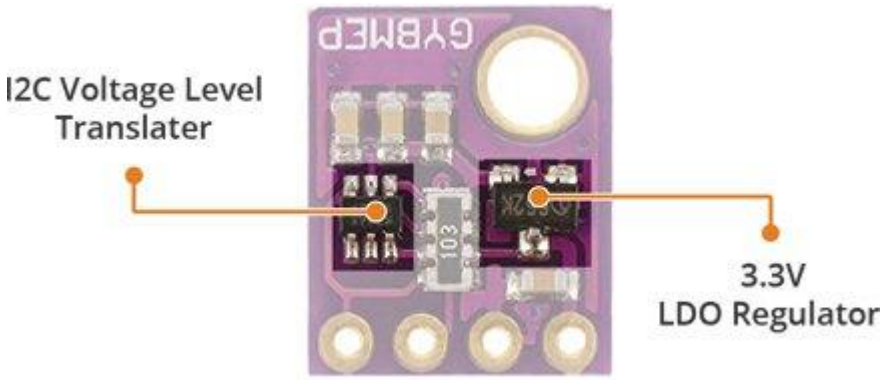



Рис.3.6. Регулятор напруги 3,3 В LM6206 і $I^2C$ перетворювач рівнів сигналів розміщені на платі модуля

Сенсор BME280 споживає менше 1 мА при вимірюваннях і лише 5 мкА в режимі очікування. Таке низьке енергоспоживання дозволяє розробляти портативні пристрої з автономним живленням від батарейок або акумуляторів, такі як GPS модулі або годинники тощо.

**$I^2C$ інтерфейс**

Модуль має простий двохпровідний інтерфейс $I^2C$, що дає змогу легко підключити будь-який мікроконтролер. $I^2C$ адрес модуля BME280 за замовчуванням є 0×76 (Hex) і може бути легко змінений на 0×77 (Hex) за допомогою джампера.

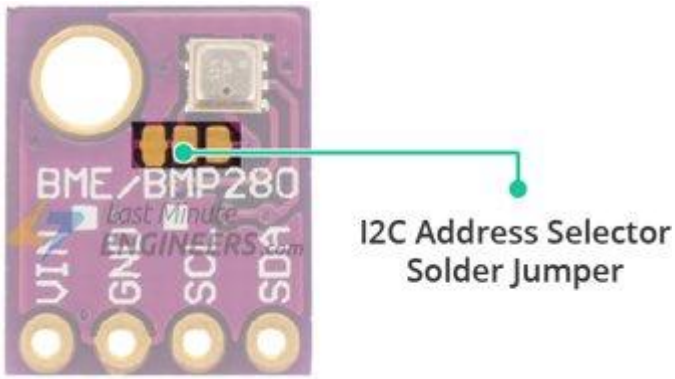



Рис.3.7. Джампер для вибору $I^2C$ адреси сенсора BME280

**Процедура зміни $I^2C$ адреси є наступною:**

1. Шукаємо перемичку (джампер) з припою розміщену поза чіпом. За замовчуванням середня мідна контактна площадка з'єднана з лівою контактною площадкою.
2. Здираємо з'єднання (контакт) між середньою і лівою мідною контактною площадкою за допомогою гострого ножа та роз'єднуємо їх.
3. Додаємо каплю припою між середньою і правою мідними площадками, щоб їх з'єднати. Таким чином встановлюємо I2C адрес 0×77 (Hex).

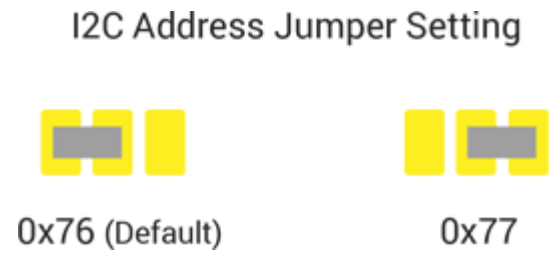



Рис.3.7. Виставлення джампера вибору $I^2C$ адреси

**Позначення та призначення виводів сенсора BME280**

Модуль BME280 має 4 піни (виводи) для підключення до мікроконтролера. Позначення виводів модуля сенсора BME280 зображено на Рис.3.8.

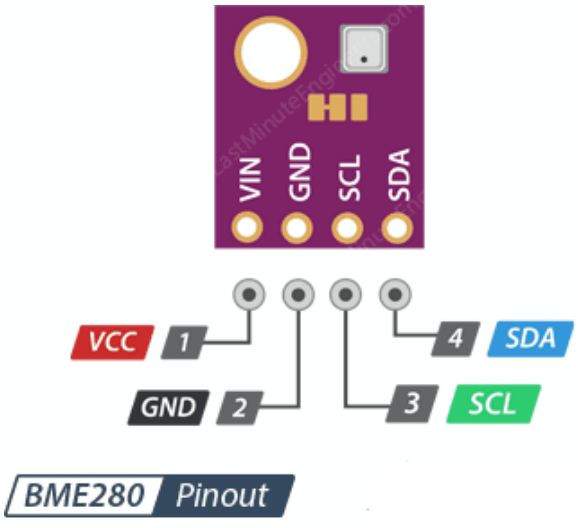



Рис.3.8. Розпіновка модуля сенсора BME280

VIN - вивід живлення модуля, яке може бути між 3,3 В і 5 В.

GND – вивід землі. Підключається до землі плати Arduino.

SCL – вивід синхронізації (тактовий) для інтерфейсу $I^2C$.

SDA – вивід послідовних даних для інтерфейсу $I^2C$.

**Підключення модуля BME280 до плати Arduino**

Підключення модуля BME280 до плати Arduino є досить простим. Підключаємо вивід VIN до 5 В, яке можемо взяти з плати або зовнішнього модуля джерела живлення. Вивід GND з'єднуємо з землею. Тепер залишилося підключити піни (виводи), які використовуються для підключення по інтерфейсу $I^2C$. Слід відзначити, що кожний тип плат Arduino використовує різні виводи для роботи по інтерфейсу $I^2C$. На платах Arduino Uno R3, SDA (лінія передачі даних) і SCL (лінія передачі синхросигнвлу (синхронізації)) є на контактному роз'ємі близько до піна AREF – це піни A5 (SCL) і A4 (SDA). В платі Arduino Mega для підключення по інтерфейсу $I^2C$ призначені цифрові виводи 21 (SCL) і 20 (SDA). В Табл.3.1 наведено виводи інтерфейсу $I^2C$ для різних плат Arduino.

Табл.3.1. Виводи інтерфейсу $I^2C$ для різних плат Arduino

| Тип плати Arduino | SCL | SDA |
|---|---|---|
| Arduino Uno | A5 | A4 |
| Arduino Nano | A5 | A4 |
| Arduino Mega | 21 | 20 |
| Leonardo/Micro | 3 | 2 |

На Рис.3.9 зображено схему підключення модуля BME280 до МК плати Arduino Mega2560 по інтерфейсу $I^2C$.

**Підключення до плати Arduino**

Сенсор підтримує два інтерфейси – $I^2C$ і SPI, тому модуль можна підключати двома способами. Для підключення по інтерфейсу $I^2C$ використовуються два виводи Arduino.

| BME280 | Arduino Uno | Arduino Mega |
|---|---|---|

| | | |
|---|---|---|
| GND | GND | GND |
| VCC | +3,3 | +3,3 |
| SDA | A4 | 20 |
| SCL | A5 | 21 |

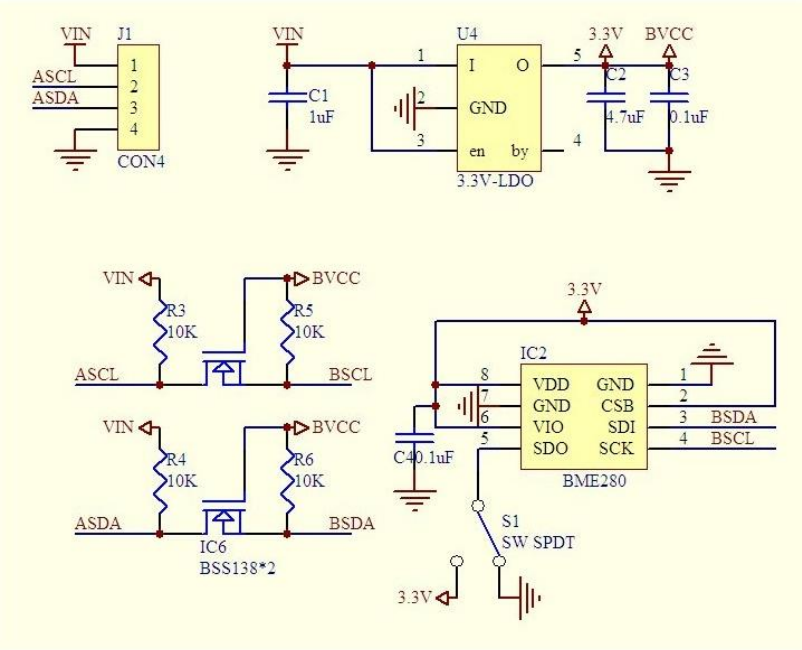



Рис.3.9. Підключення модуля BME280 до МК по інтерфейсу $I^2C$

Для підключення по протоколу SPI використовуються 4 виводи Arduino.

Табл.3.2. Виводи підключення модуля BME280 по інтерфейсу SPI для плат Arduino Uno і Mega

| BME280 | Arduino Uno | Arduino Mega |
|---|---|---|
| GND | GND | GND |
| VCC | +3,3 В | +3,3 В |
| SCK | D13 | D52 |
| CSB | D10 | D49 |
| SDI | D11 | D51 |
| SDO | D12 | D50 |

**Опис Wi-Fi модуля ESP-01 на чіпі ESP8266**

ESP-01 – це плата-модуль Wi-Fi на базі популярного чіпсета ESP8266EX. На платі знаходиться мікросхема Flash-пам’яті об’ємом 2 МБ, чіп ESP8266EX, кварцовий резонатор, два індикаторних світлодіода і мініатюрна антена з

доріжки на верхньому шарі друкованої плати у вигляді змійки. Flash-пам'ять необхідна для зберігання програмного забезпечення. При кожному включенні живлення, ПЗ автоматично завантажується в чіп ESP8266EX.

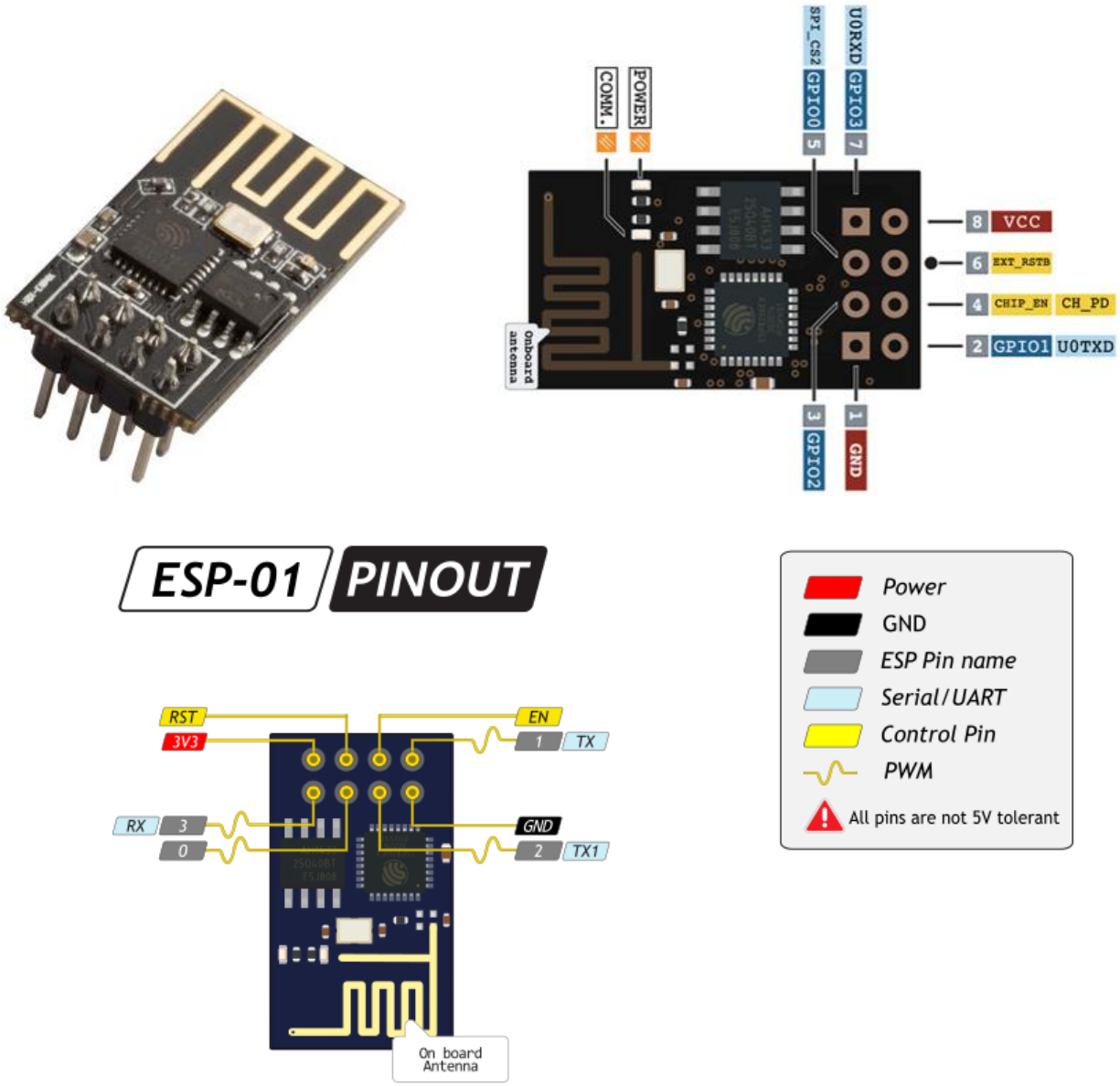



Рис.3.10. Wi-Fi модуль ESP-01 на чіпі ESP8266

На виводи живлення 3V3 і GND (Рис. 3.10) можна подавати від 3 до 3,6 В. Модуль базується на 3,3 В логіці і в піках може споживати до 200 мА. Оскільки майже жодна з плат Arduino не може витримати таку нагрузку, тому модуль ESP-01 потрібно живити від зовнішнього джерела живлення. Вивід 6 RST/EXT_RSTB модуля ESP-01 призначений для перезавантаження модуля, короткочасно подавши на нього низький логічний рівень, модуль

перезавантажитися. Вивід RST підключають через резистор 10 кОм до плюса живлення, щоб забезпечити кращу стабільність роботи модуля. Вивід 4 CH_PD/CHIP_EN служить для переведення модуля в енергозберігаючий режим, в якому він споживає дуже малий струм. Цей вивід також рекондовано підключити через резистор на 10 кОм до плюса. Виводи RX/U0RXD 7 TX/U0TXD 2 апаратний UART використовуються для підключення до МК і для перепрошивки. Також вони можуть використовуватися як порти вводу/виводу GPIO (GPIO3 і GPIO1 відповідно).

***Характеристики Wi-Fi модуля ESP-01:***

- Модуль: ESP-01 з чіпом ESP8266EX;
- вихідний інтерфейс: UART;
- об’єм Flash-пам’яті: 2 МБ;
- безпровідний інтерфейс: Wi-Fi 802.11 b/g/n 2,4 ГГц;
- режими роботи:
  - клієнт (STA);
  - точка доступу (AP);
  - клієнт + Точка доступу (STA + AP);
- напруга живлення: 3,3 В
- струм споживання: до 250 мА
- габарити: 25×15 мм

**Чіп ESP8266EX**

Чіп ESP8266 виконаний за технологією SoC (System-on-a-Chip - система на кристалі). В кристал входить процесор сімейства Xtensa - 32-бітний Tensilica L106 з частотою 80 МГц з ультранизьким енергоспоживанням, радіочастотний прийомопередавач (трансівер) з фізичним рівнем WiFi IEEE 802.11 b/g і блоки пам’яті SRAM. Потужності процесорного ядра вистачає для роботи складних користувацьких програм і цифрової сигнальної обробки.

Програма користувача повинна зберігатися на зовнішній мікросхемі Flash-пам’яті і завантажуватися в ESP8266EX через один з доступних інтерфейсів (SPI, UART, SDIO та ін.) кожний раз в момент включення живлення системи.

Чіп ESP8266 не містить в собі Flash-пам’ять та багатьох інших компонентів для користувацького старту. Мікросхема є основою на базі якої випускаються модулі з необхідною периферією, наприклад ESP-01.

**Робота з АТ-командами. Підключення і настройка.**

За замовчуванням модуль налаштований на роботу через АТ-команди. Управляюча плата посилає команди - Wi-Fi модуль виконує відповідну операцію. У стандартній прошивці Wi-Fi модуль спілкується з управляючою платою через АТ-команди по протоколу UART. На всіх платах Arduino присутній хоча б один апаратний UART - HardwareSerial. Якщо ж з якихось причин він зайнятий іншим пристроєм, можна скористатися програмним UART - SoftwareSerial.

***HardwareSerial Mega***

На платах форм-фактору Arduino Mega 2560 апаратний UART, який відповідає за передачу даних через піни 1 і 0, відповідає також за передачу по USB. Це означає неможливість використовувати одночасно UART для комунікації з Wi-Fi модулем і відлагодження по USB.

На платах такого форм-фактору є ще додатково три апаратних UART:

- Serial1: Піни 19 (RX1) і 18 (TX1);
- Serial2: Піни 17 (RX2) і 16 (TX2);
- Serial3: Піни 15 (RX3) і 14 (TX3).

Підключаємо Wi-Fi модуль до об’єкта Serial2 на піни 16 і 17 на платі Arduino Mega2560.

Табл.3.3. АТ - команди Wi-Fi модуля ESP-01

| Функція | AT-команда | Відповідь |
|---|---|---|
| Робочий | AT | OK |
| Перезавантаження | AT+RST | OK [System Ready, Vendor:www.ai-thinker.com] |
| Версія прошивки | AT+GMR | AT+GMR 0018000902 OK |
| Список точок доступу | AT+CWLAP | AT+CWLAP<br>+CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1)<br>+CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1)<br>OK |
| Підключитися до точки доступу | AT+CWJAP?<br>AT+CWJAP="SSID","Password" | Query AT+CWJAP?<br>+CWJAP:"RochefortSurLac"<br>OK |
| Відключитися від точки доступу | AT+CWQAP=?<br>AT+CWQAP | Query<br>OK |
| Отримати IP адрес | AT+CIFSR | AT+CIFSR 192.168.0.105<br>OK |
| Встановити параметри точки доступу | AT+ CWSAP?<br>AT+ CWSAP=<br><ssid>,<pwd>,<chl>,<ecn> | Query<br>ssid, pwd<br>chl = channel, ecn = encryption |
| WiFi Режим | AT+CWMODE?<br>AT+CWMODE=1<br>AT+CWMODE=2<br>AT+CWMODE=3 | Query<br>STA<br>AP<br>BOTH |
| Налаштувати TCP або UDP підключення | AT+CIPSTART=?<br>(CIPMUX=0) AT+CIPSTART =<br><type>,<addr>,<port><br>(CIPMUX=1) AT+CIPSTART=<br><id><type>,<addr>, <port> | Query<br>id = 0-4, type = TCP/UDP, addr = IP address, port= port |
| TCP/UDP підключення | AT+ CIPMUX?<br>AT+ CIPMUX=0<br>AT+ CIPMUX=1 | Query<br>Single<br>Multiple |
| Перевірити IP підключених пристроїв | AT+CWLIF | |
| Статус TCP/IP підключення | AT+CIPSTATUS | AT+CIPSTATUS? no this fun |
| Відіслати TCP/IP дані | (CIPMUX=0)<br>AT+CIPSEND=<length>;<br>(CIPMUX=1) AT+CIPSEND=<br><id>,<length> | |
| Закрити TCP/UDP підключення | AT+CIPCLOSE=<id> або<br>AT+CIPCLOSE | |
| Встановити як сервер | AT+ CIPSERVER=<br><mode>[,<port>] | mode 0 закрити режим сервера; mod<br>1 відкрити; port = port |
| Налаштувати час очікування сервера | AT+CIPSTO?<br>AT+CIPSTO=<time> | Query<br><time>0~28800 в секундах |
| Швидкість передачі даних | AT+CIOBAUD?<br>підтримує: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600 | Query AT+CIOBAUD?<br>+CIOBAUD:9600 OK |

**Алфавітно-цифровий РК-модуль на основі контролера HD44780**

Контролер HD44780 фірми Hitachi є промисловим стандартом і широко використовується у виготовлені алфавітно-цифрових РК-модулів. Аналоги цього контролера або сумісні з ним по інтерфейсу і командній мові мікросхеми, випускають багато фірм, серед них: Epson, Sanyo, Samsung, Philips. Ще більше число фірм виготовляють РК-модулі на базі цих контролерів. Ці модулі можна зустріти в найрізноманітніших пристроях: вимірювальні прилади, медичне обладнання, промислове і технологічне обладнання, офісна техніка – принтери, телефони, факсові та копіювальні апарати.

Алфавітно-цифрові РК-модулі представляють собою недороге і зручне рішення, що дозволяє зекономити час і ресурси при розробці нових виробів, при цьому забезпечують відображення великого об'єму інформації при хорошій читабельності і низькому енергоспоживанню. Наявність в РК-модулях задньої підсвітки дозволяє експлуатувати їх в умовах з малим або нульовим освітленням, а виконання з розширеним діапазоном температур (-20 ºС…+70 ºС) в складних експлуатаційних умовах, в тому числі в переносній, польовій і навіть, інколи, у бортовій апаратурі.

Контролер HD44780 потенційно може управляти (керувати) 2-ма рядками по 40 символів в кожному (для модулів з 4-ма рядками по 40 символів використовуються два однотипних контролера), при матриці символу 5×7 пікселів. Контролер також підтримує символи з матрицею 5×10 пікселів.

Існує декілька різних більш-менш стандартних форматів РК-модулів (символів×рядків): 8×2, 16×1, 16×2, 16×4, 20×1, 20×2, 20×4, 24×2, 40×2, 40×4. Зустрічаються і менш поширені формати: 8×1, 12×2, 32×2 та інші – принципових обмежень на комбінації і кількість відображувальних символів контролер не накладає – модуль може мати будь-яку кількість символів від 10 до 80, хоча в деяких комбінаціях програмна адресація символів може виявитися (бути) не дуже зручною.

В рамках одного формату можуть виготовлятися РК-модулі декількох конструктивів, які відрізняються габаритами РКІ (і, як результат, розмірами

символів), так і розмірами плати і посадки. Наприклад, фірма Powertip пропонує алфавітно-цифрові РК-модулі 11-ти форматів (від 8×2 до 40×4) в 37-ми різних конструктивах, 16×1 в 6-ти, а модулі формату 16×2 в 11-ти.

Вивчаючи каталоги різних фірм-виробників РК-модулів, можна переконатися, що одні формати і конструктиви являються власними розробками і не мають аналогів в номенклатурі решти фірм, інші являються фактично стандартами і виготовляються більшістю виробників. Як приклад, можна назвати РК-модулі формату 24×2, названий PC2402-A у Powertip, ED24200 у EDT, DMC-24227 в Optrex, SC2402A в Bolymin, MDI-S-24265 у Veritronix, PVC240202 в Pievue та інші. Всі ці модулі мають одинакові конструктивні розміри і являються взаємозамінними.

В рамках одного конструктиву РК-модуль може мати ще ряд модифікацій. Зокрема, можуть використовуватися декілька типів РКІ, які відрізняються кольором фону і коліром символів, а також по використанню РК-матеріалів і структурі: TN. STN і FSTN типу. РКІ STN і FSTN дорожчі і мають підвищену контрастність і вдвоє більший кут огляду, причому РКІ FSTN типу мають кращі характеристики ніж STN.

РК-модулі можуть мати задню підсвітку, яка розміщена між РКІ і друкованою платою. Тому РКІ виготовляються з напівпрозорим і прозорим заднім шаром (в останньому випадку зчитування інформації можливе лише при наявності підсвітки). Власне підсвітка може бути реалізована декількома способами: за допомогою електролюмінісцентної панелі, яка представляє собою тонку плівку, що випромінює світло при прикладенні змінного струму підвищеної напруги порядку 100…150 В; люмінісцентної лампи з холодним католом (також працює при підвищеній напрузі), випромінювання якої рівномірно розподіляється по всій площі РКІ за допомогою відбивача або плоского світловоду; третій варіант – підсвітка на основі світлодіодної матриці.

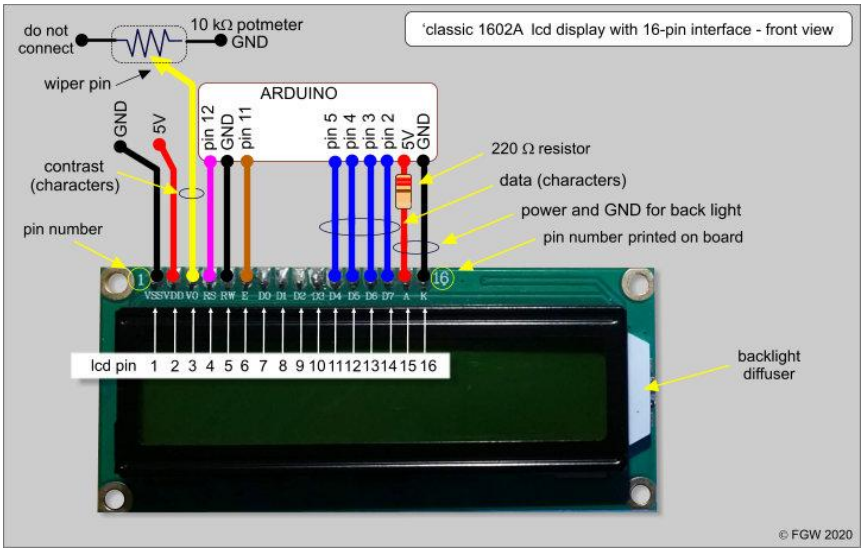
Перші два способи підсвітки забезпечують високу яскравість і можуть мати білий тон свічення при відносно низькому енергоспоживанні, але потребують джерело підвищеної напруги, що створює деякі труднощі при створенні

апаратури з автономним живленням. Навпаки, світлодіодна підсвітка не потребує високовольтного джерела живлення (пряме падіння напруги складає 4,2 В) і при використанні нескладного джерела струму дозволить виробляти живлення від джерела з напругою 5 В. Крім того, світлодіодна підсвітка має значно більший (в десятки разів) час напрацювання, а також тільки вона допустима до експлуатації в розширеному діапазоні температур (-20 ºС…+70 ºС).

**Підключення алфавітно-цифрового РК-модуля HD44780 до МК плати Arduino Mega2560**

Для підключення РК-модуля з управляючою системою використовується паралельна синхронна шина, яка має 8 або 4 (вибирається програмно) ліній даних DB0…DB7, лінію вибору операції R/W, лінію вибору регістра RS і лінію стробування/синхронізації E. Крім ліній шини управління є дві лінії для подачі напруги живлення 5 В – GND і Vcc, і лінія для подачі напруги живлення драйвера РКІ – Vo.


Рис.3.10. РК - модуль 16×2



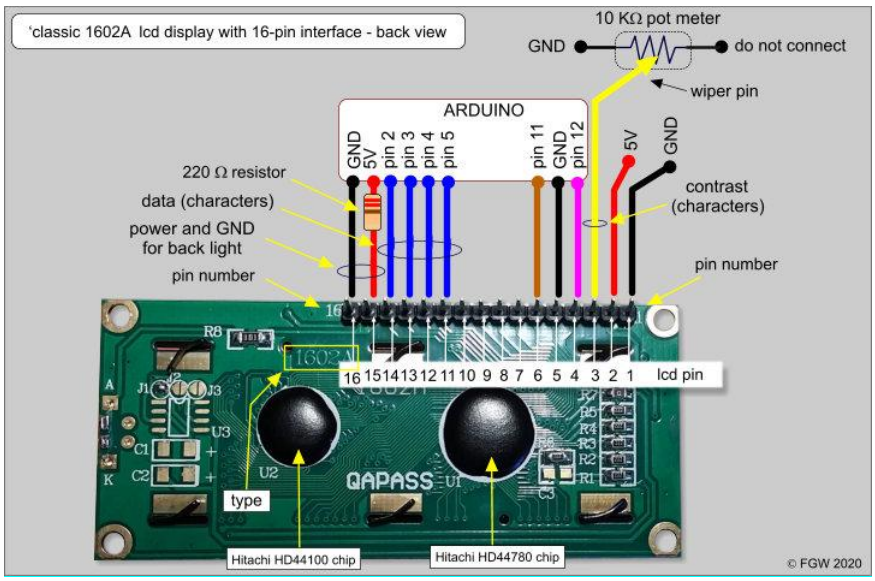



Рис.3.11. Призначення виводів та підключення РК-модуля до Arduino

Для підключення LCD використано 4-бітний режим. Виводи LCD підключено до виводів плати Arduino Mega2560 в наступному порядку:

- LCD пін (вивід) RS до цифрового піна 2 плати Arduino Mega2560;
- LCD пін Enable до цифрового піна 3 плати Arduino Mega2560;
- LCD пін даних D4 до цифрового піна 4 плати Arduino Mega2560;
- LCD пін даних D5 до цифрового піна 5 плати Arduino Mega2560;
- LCD пін даних D6 до цифрового піна 6 плати Arduino Mega2560;
- LCD пін даних D7 до цифрового піна 7 плати Arduino Mega2560.

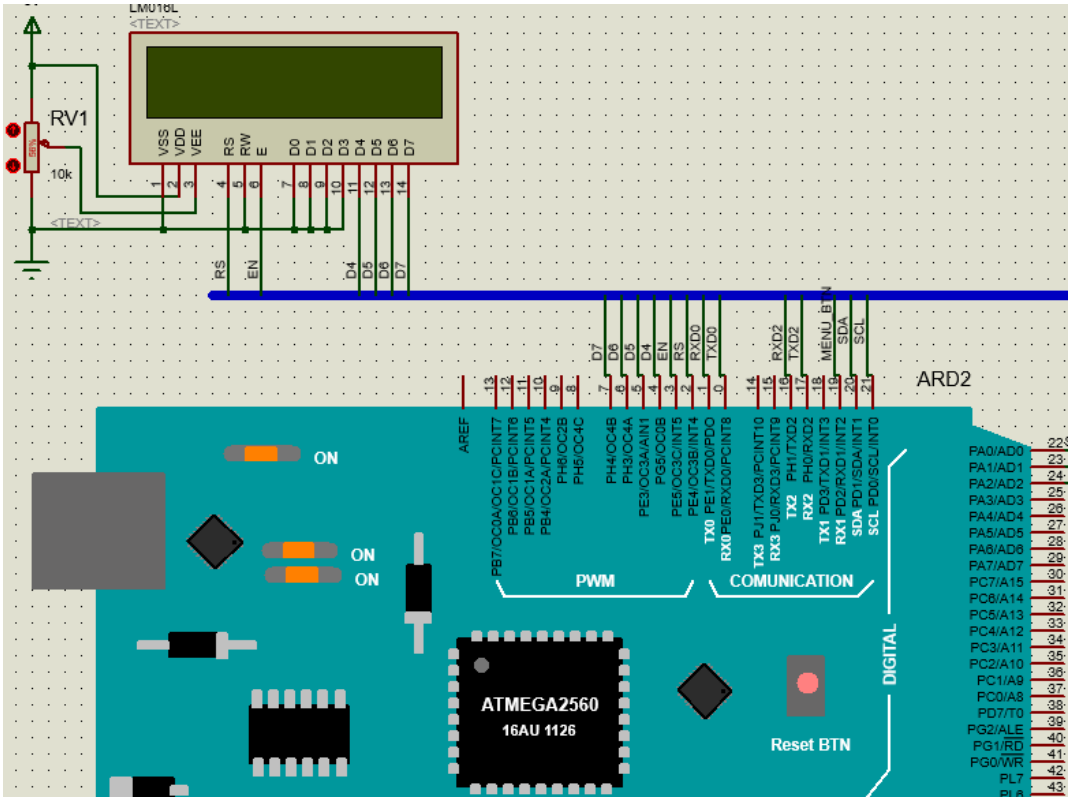



Рис.3.12. Підключення LCD 16×2 до плати Arduino Mega2560

**IC 16×2 точково-матричного LCD контролера HD44780**

Для того щоб відобразити свій власний символ потрібно якось вказати IC як цей власний (користувацький) символ має виглядати. Всередині IC LCD контролера HD44780 є три типи пам'яті:

1. **ROM генератора символів (CGROM)**: Це постійна пам'ять лише для читання і призначена для зберігання всіх шаблонів символів, що наперед визначені всередині неї.
2. **RAM даних дисплею (DDRAM)**: Це є ОЗП. Кожний раз при виведені символа його шаблон витягується (зчитується) з CGROM і передається в DRAM, а потім виводиться на екран дисплею. Щоб спростити вивід символа на екран, DDRAM містить шаблони всіх символів, які в даний момент відображаються на екрані LCD.
3. **ОЗП (RAM) генератора символів (CGRAM):** це оперативна пам'ять для запису і читання даних. Пам'ять використовується для генерування користувацького або власного символу. Потрібно створити шаблон символу і записати його у CGRAM.

В datasheet на контролер HD44780 зазначено, що IC HD44780 має 8 позицій для зберігання власних шаблонів символів користувача у CGRAM, а також визначенх наперед символів, які можна відобразити на екрані LCD. Розмір CGRAM становить 64 байти. В CGRAM одночасно можна згенерувати/розмістити до 8 символів розміром 5×8 пікселів. Можна також в CGRAM розмістити символи розміром 5×10 пікселів, але тільки 4.

Першим кроком є генерування шаблону свого власного символу. Кожний символ є комбінацією 5×8 точок (пікселів). Потрібно вибрати яка точка (піксель) має бути засвіченим, а який залищатися не засвіченим. Для створення власного символу можна використати веб – аплікацією https://maxpromer.github.io/LCD-Character-Creator/ або спеціальну програму, таку як MikroElektronika GLCD Font Creator 1.2.0.0.

Для створення символу є (піксельна матриця 5×8) сітка розмірністю 5×8 клітинок (пікселів), в якій відмічають пікселі, що мають бути засвічені. Програма автоматично згенеровує код програми зі створеним шаблоном створеного символу у вигляді байтового масиву в двійковій або шістнадцятковій системі (Binary або Hex - байтовий масив). Наприклад, створені шаблони символів (значків) температури, вологості і тиску та згенерований програмний код для Arduino та AVR в LCD Character Creator і MikroElektronika GLCD Font Creator зображено на Рис.3.13 – 3.20.

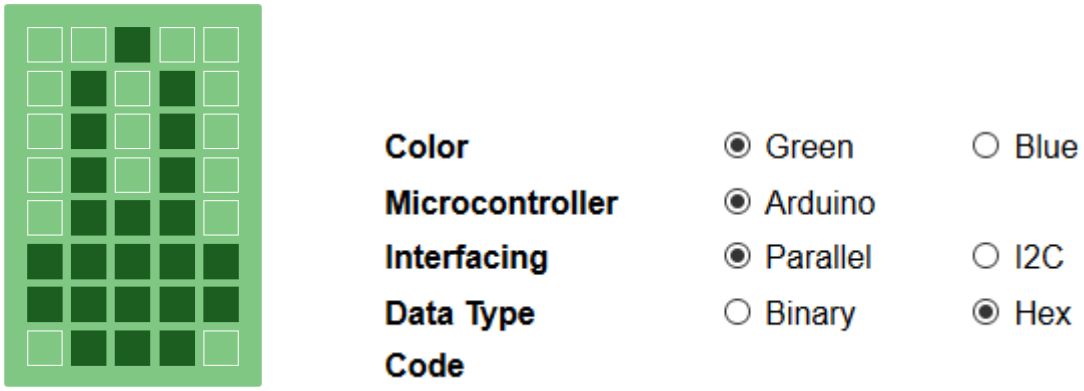
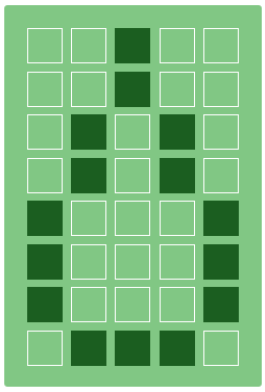



Рис.3.13. Створений шаблон LCD значка (символа) температури за допомогою веб-аплікації LCD Custom Character Generator

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // RS, E, D4, D5, D6, D7

byte customChar[] = {
  0x04, /* B00100 */
  0x0A, /* B01010 */
  0x0A, /* B01010 */
  0x0A, /* B01010 */
  0x0E, /* B01110 */
  0x1F, /* B11111 */
  0x1F, /* B11111 */
  0x0E  /* B01110 */
};

void setup() {
  lcd.begin(16, 2);
  lcd.createChar(0, customChar);
  lcd.home();
  lcd.write(0);
}

void loop() { }
```

Рис.3.14. Код згенерований для Arduino зі шаблоном LCD значка (символа) температури за допомогою веб-аплікації LCD Custom Character Generator



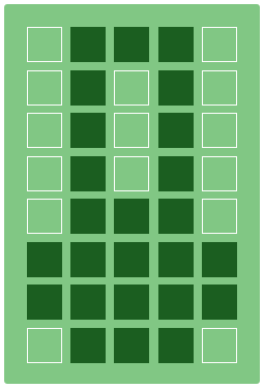
| | | |
|---|---|---|
| **Color** | ◉ Green | ○ Blue |
| **Microcontroller** | ◉ Arduino | |
| **Interfacing** | ◉ Parallel | ○ I2C |
| **Data Type** | ○ Binary | ◉ Hex |
| **Code** | | |

Рис.3.15. Створений шаблон LCD значка (символа) вологості за допомогою веб-аплікації LCD Custom Character Generator

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // RS, E, D4, D5, D6, D7

byte customChar[] = {
  0x04, /* B00100 */
  0x04, /* B00100 */
  0x0A, /* B01010 */
  0x0A, /* B01010 */
  0x11, /* B10001 */
  0x11, /* B10001 */
  0x11, /* B10001 */
  0x0E  /* B01110 */
};

void setup() {
```

```
  lcd.begin(16, 2);
  lcd.createChar(0, customChar);
  lcd.home();
  lcd.write(0);
}

void loop() { }
```

Рис.3.16. Код згенерований для Arduino зі шаблоном LCD значка (символа) вологості за допомогою веб-аплікації LCD Custom Character Generator



| | | |
|---|---|---|
| **Color** | ◉ Green | ○ Blue |
| **Microcontroller** | ◉ Arduino | |
| **Interfacing** | ◉ Parallel | ○ I2C |
| **Data Type** | ○ Binary | ◉ Hex |
| **Code** | | |

Рис.3.17. Створений шаблон LCD значка (символа) тиску за допомогою веб-аплікації LCD Custom Character Generator

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // RS, E, D4, D5, D6, D7

byte customChar[] = {
  0x0E, /* B01110 */
  0x0A, /* B01010 */
  0x0A, /* B01010 */
  0x0A, /* B01010 */
  0x0E, /* B01110 */
  0x1F, /* B11111 */
  0x1F, /* B11111 */
  0x0E  /* B01110 */
};

void setup() {
  lcd.begin(16, 2);
  lcd.createChar(0, customChar);
  lcd.home();
  lcd.write(0);
}

void loop() { }
```

Рис.3.18. Код згенерований для Arduino зі шаблоном LCD значка (символа) тиску за допомогою веб-аплікації LCD Custom Character Generator

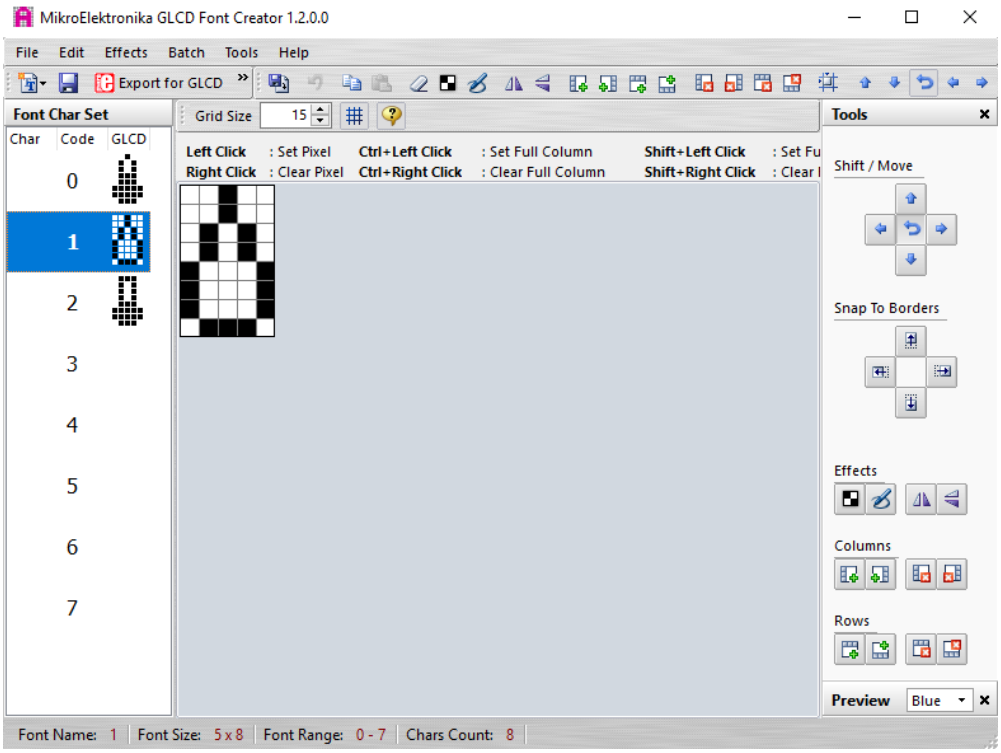



Рис.3.19. Створення власного шаблону символа в програмі MikroElektronika GLCD Font Creator

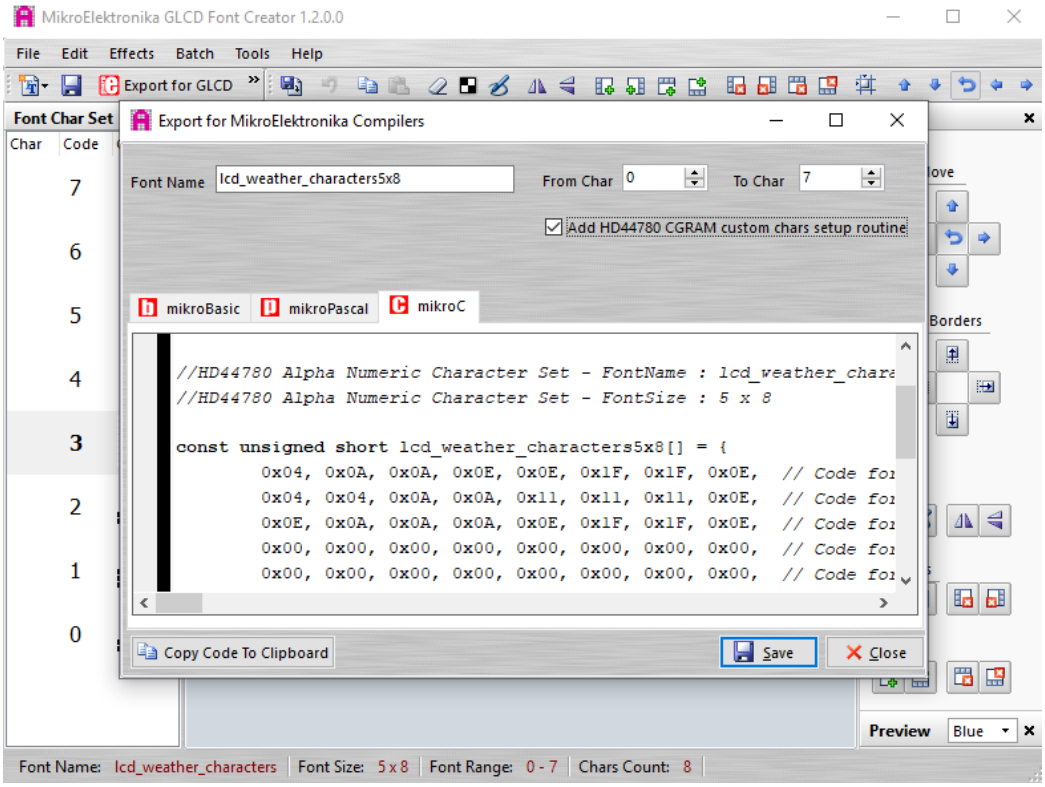



Рис.3.20. Програмний код зі створеними шаблонами символів у вигляді Hex - масиву lcd_characters5x8 згенерований програмою MikroElektronika GLCD Font Creator

**Модуль живлення для макетної плати 5V/3.3V від RobotDyn**

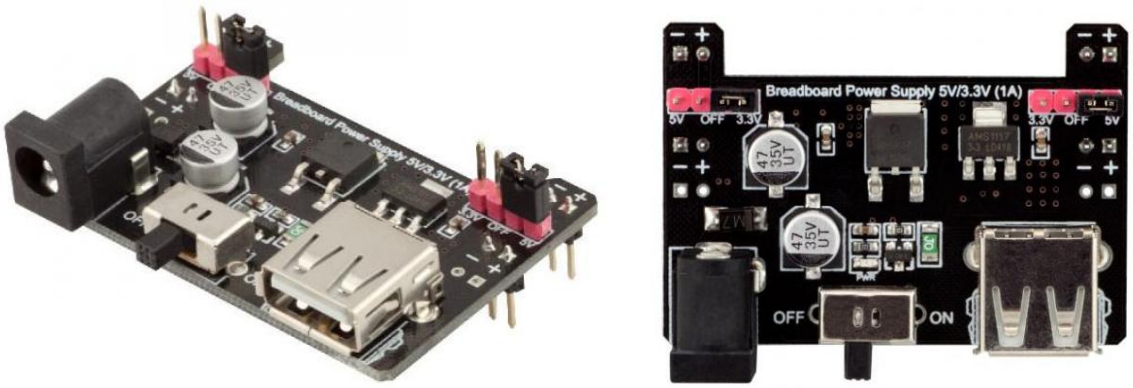



Рис.3.21. Модуль живлення для макетної плати 5V/3.3V

Високоякісний модуль живлення для макетної плати призначений для живлення безпаєчних макетних плат, які мають по бокам дві лінії живлення. Модуль має перемикач 3.3V/5V. Живеться модуль від зовнішнього блоку живлення 6 - 15 В.

**Характеристики модуля:**

- вхідна напруга: DC 6~15 В;
- вихідна напруга на USB: 5 В;
- вихідна напруга на контактах: 3.3В/5В;
- вихідний струм на USB: 500 мА
- Вихідний струм на контактах: 1 А – 5 В / 800 мА - 3.3 В;
- Захист від переполюсування (переплутання полюсів).

## 3.2. Проектування апаратного забезпечення IoT – пристрою моніторингу метеопараметрів засобами САПР Proteus

IoT – пристрій повинен моніторити такі параметри погоди (метеопараметри) як: температура, відносна вологість повітря, атмосферний тиск. На Рис.3.22 зображено структуру АЗ IoT – системи моніторингу метеопарметрів. Апаратне забезпечення IoT - пристрою моніторингу метеопарметрів розроблене на платформі Arduino Mega2560 з мікроконтролером AVR ATmega2560. До МК плати Arduino Mega2560 підключено цифровий сенсор тиску, температури і

вологості BME280, Wi-Fi модуль ESP8266, модуль алфавітно-цифрового LCD на контролері Hitachi HD44780, 4 кнопки. Мікроконтролер ATmega2560 плати Arduino Mega2560 зчитує та обробляє метеодані з сенсора BME280. Поточні значення метеопараметрів пристрій виводить на алфавітно-цифровий 16×2 LCD.

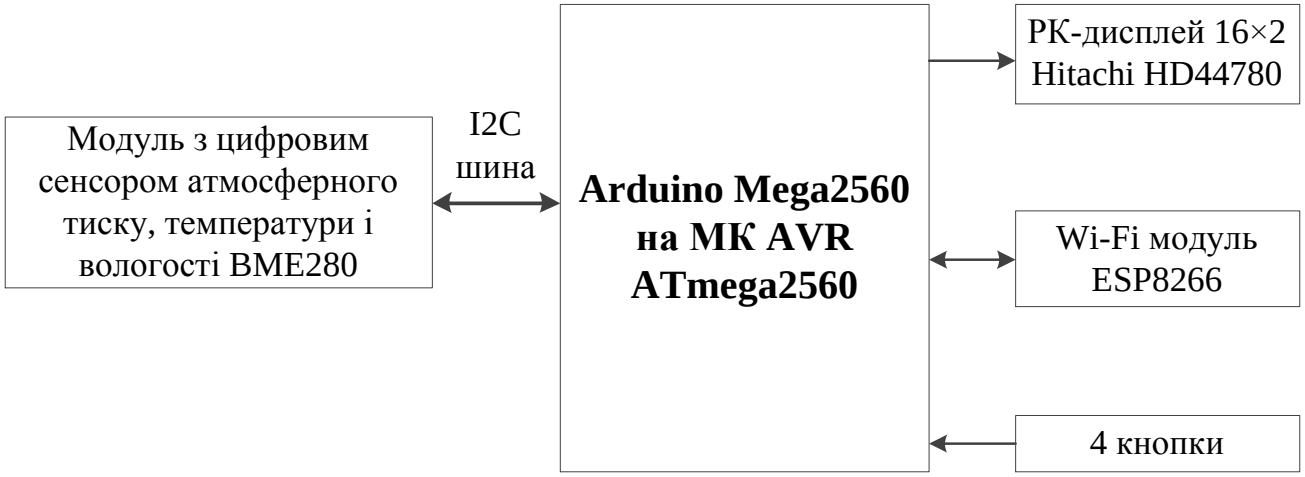



Рис.3.22. Загальна структура апаратного забезпечення IoT – пристрою моніторингу метеопараметрів

На Рис.3.23 зображено спроектоване АЗ пристрою в САПР Proteus. На схемі цифровий сенсор атмосферного тиску, температури і вологості BME280 підключено по шині $I^2C$ до виводів 20 (SDA/INT1) і 21 (SCL/INT0) плати Arduino Mega2560. Wi-Fi модуль ESP8266 піни ESP_RX і EXP_TX підключено по інтерфейсу UART до пінів 16 (TX2) і 17 (RX2) плати Arduino Mega2560. Шістнадцяти символьний двохрядковий РК-дисплей на базі контролера Hitachi HD44780 підключено до виводів 2…7 (PE0/PCINT8/RXD0, PE1/TXD0, PE4/INT4/OC3B, PE5/INT5/OC3C, PG5/OC0B, PE3/AINT1/OC3A, PH3/OC4A, PH4/OC4B) плати Arduino Mega2560. Вивід вибору регістра (RS – register select) РК-дисплею з'єднано з цифровим піном 2 плати Arduino Mega2560. Вивід стропування/синхронізації (E - enable) РК-дисплею з'єднано з цифровим піном 3 плати Arduino Mega2560. Виводи даних D4…D7 РК-дисплею з'єднано з цифровими пінами 4…7 плати Arduino Mega2560, відповідно. До виводу VEE

РК-дисплею підключено потенціометр RV1 10 кОм за допомогою якого можна подавати напругу від 0 до 5 В для регулювання контрасту дисплея.

Кнопка MENU_BTN підключена до виводу 19 (PD2/INT2/RXD1), кнопка SELECT_PLUS_BTN (вибрати/збільшити) до виводу 23 (PA1/AD1), кнопка SELECT_MINUS_BTN (вибрати/зменшити) до 22 виводу (PA0/AD0) і кнопка виходу з меню в робочий режим EXIT_BTN до піна 24 (PA2/AD2) плати Arduino Mega2560.

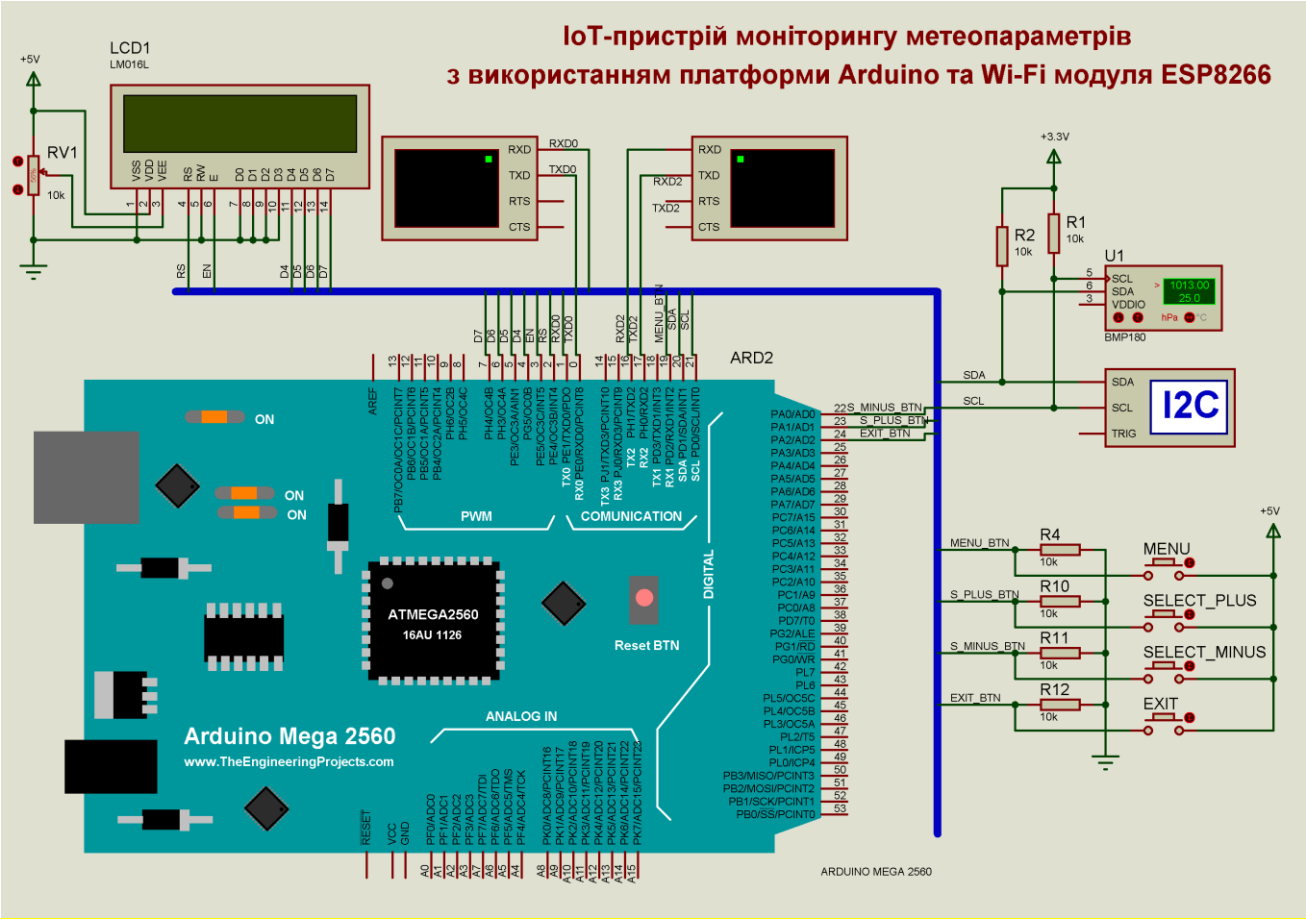



Рис.3.23. Апаратне забезпечення IoT – пристрою моніторингу метеопараметрів спроектоване в САПР Proteus

# РОЗДІЛ IV. ПРОГРАМНО-АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ IoT – ПРИСТРОЮ МОНІТОРИНГУ МЕТЕОПАРАМЕТРІВ

## 4.1. Алгоритм роботи IoT – пристрою моніторингу метеопараметрів

На Рис.4.1 зображено блок-схему алгоритму роботи IoT - пристрою моніторингу метеопараметрів.

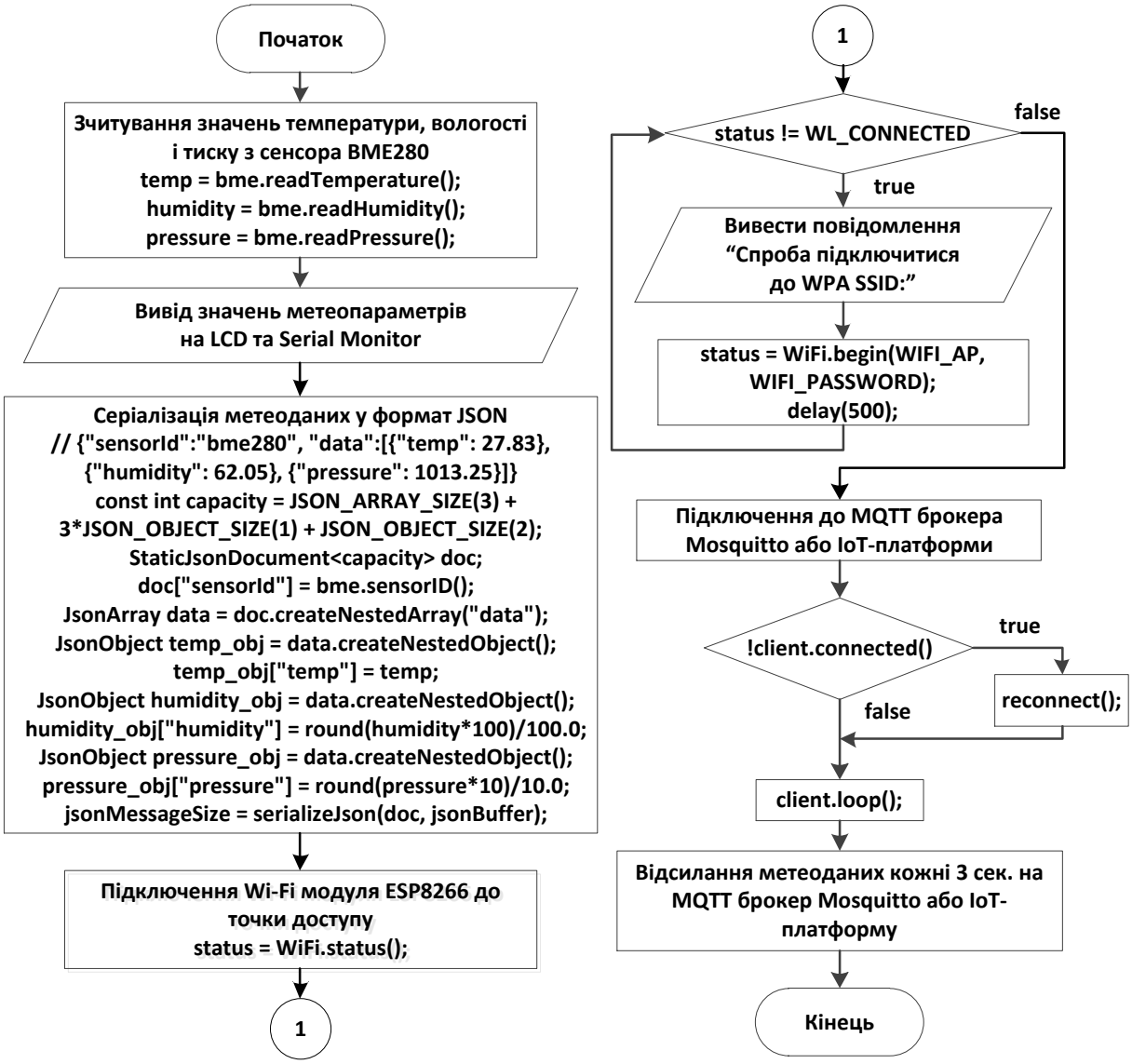



Рис. 4.1. Алгоритм роботи IoT – пристрою моніторингу метеопараметрів

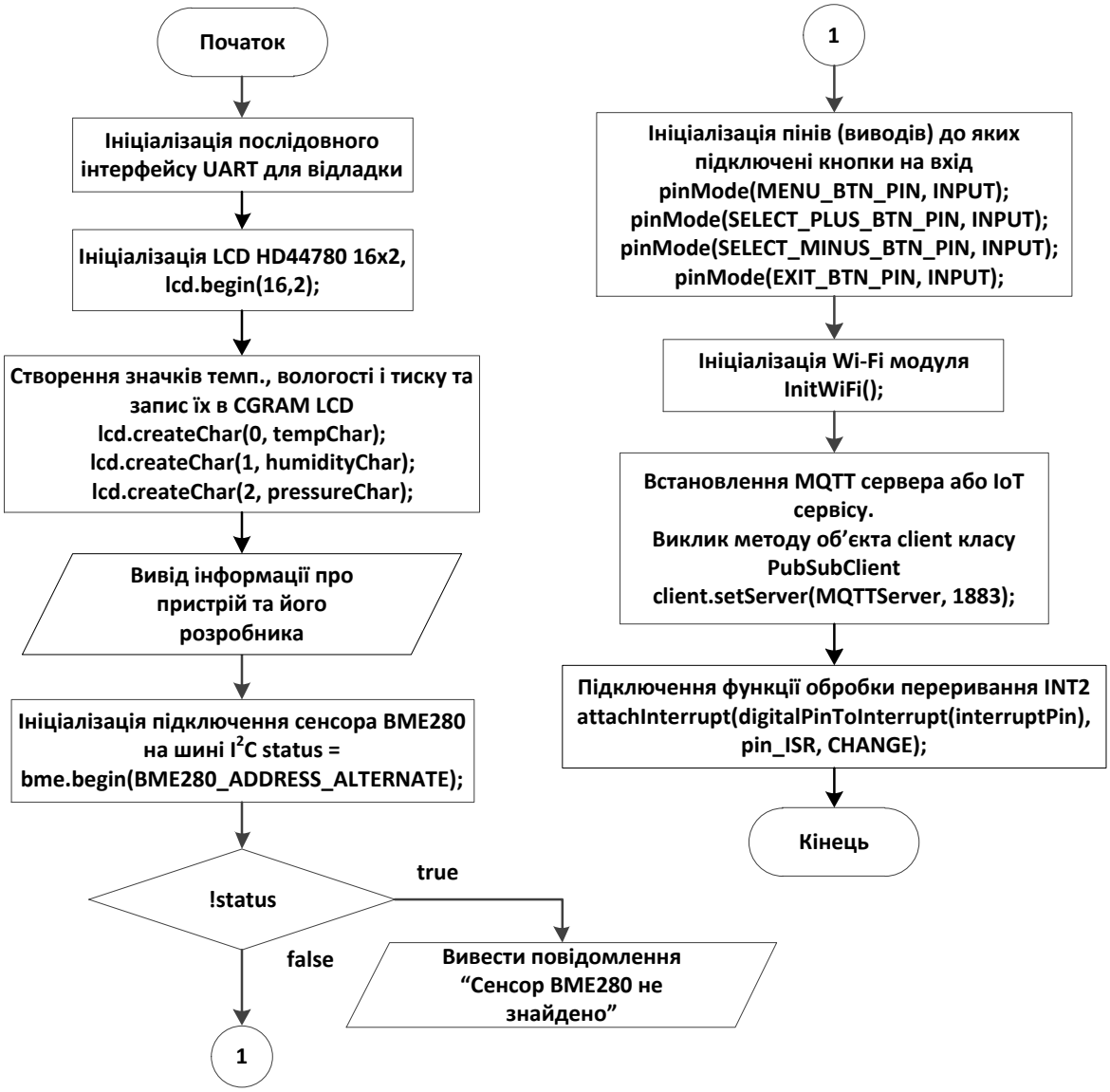



Рис.4.2. Блок-схема алгоритму ініціалізації LCD, сенсора BME280, Wi-Fi модуля та MQTT клієнта

Після подачі живлення та увімкнення IoT – пристрою моніторингу параметрів погоди стартує виконання зашитої у флеш-пам'ять програми. Спочатку програма (вилик функції setup()) ініціалізує інтерфейс РК-дисплей і визначає розміри дисплею 16 стовпців і 2 рядки, потім ініціалізує послідовне з'єднання і задає швидкість передачі даних в бітах/сек. (бодах)

```
Serial.begin(9600);
while(!Serial);  // time to get serial running
```

```
  lcd.begin(16, 2);  // виставляємо кількість стовпців і рядків LCD:
  lcd.createChar(0, tempChar);
  lcd.createChar(1, humidityChar);
  lcd.createChar(2, pressureChar);
  lcd.clear();
```

Далі проходить ініціалізація сенсора атмосферного тиску, температури і вологості BME280 підключеного по інтерфейсу $I^2C$ з адресою 0×44 виконується в наступному коді програми:

```
Serial.println(F("BME280 тест..."));
unsigned status;
// налаштування за замовчуванням. Можемо також передати об'єкт бібліотеки Wire як &Wire2)
status = bme.begin(BME280_ADDRESS_ALTERNATE);
if (!status) {
 Serial.println("Не можливо знайти робочий сенсор BME280, перевірте підлючення, адрес, ID - сенсора !");
 Serial.print("SensorID був: 0x"); Serial.println(bme.sensorID(), 16);
 Serial.print("ID 0xFF можливо означає поганий адрес, BMP 180 або BMP 085\n");
 Serial.print("ID 0x56-0x58 представляє сенсор BMP280,\n");
 Serial.print("ID 0x60 представляє сенсор BME280.\n");
 Serial.print("ID 0x61 представляє сенсор BME680.\n");
 while (1);
}
 Serial.println("пройдено");
```

Ініціалізація пінів до яких підключені кнопки MENU_BTN, SELECT_PLUS_BTN, SELECT_MINUS_BTN та EXIT_BTN на вхід:

```
// ініціалізація пінів кнопок як входи:
  pinMode(MENU_BUTTON_PIN, INPUT);
  pinMode(SELECT_PLUS_BUTTON_PIN, INPUT);
```

pinMode(SELECT_MINUS_BUTTON_PIN, INPUT);

pinMode(EXIT_BUTTON_PIN, INPUT);

Далі проходить ініціалізація Wi-Fi модуля ESP8266, як MQTT клієнта, встановлення MQTT сервера і порта до якого буде виконуватися підключення і обмін повідомленнями за допомогою методу setServer об’єкту PubSubClient.

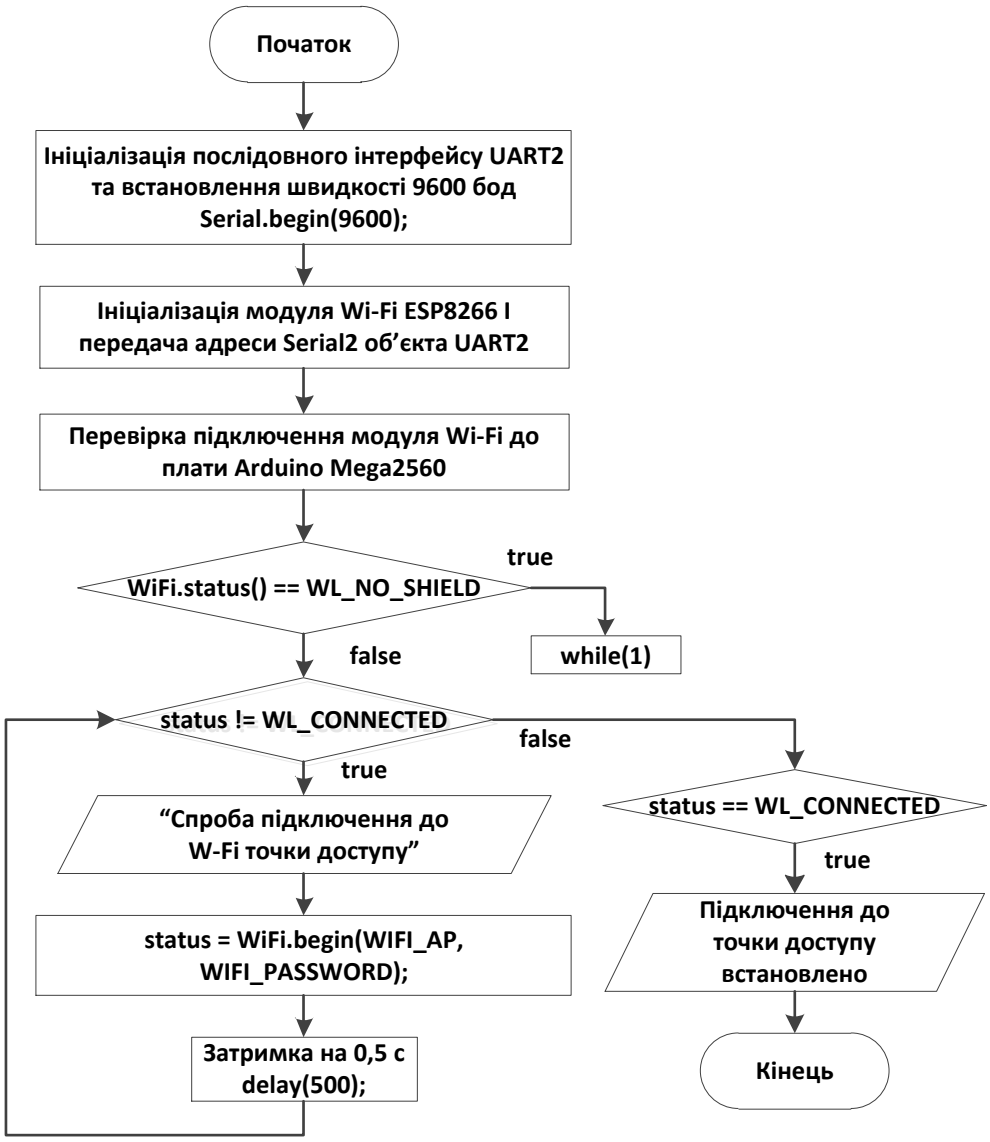



Рис.4.3. Блок-схема алгоритму ініціалізації Wi-Fi модуля ESP8266 та встановлення підключення до Wi-Fi точки доступу

// ініціалізація Wi-Fi модуля і встановлення сервера

InitWiFi();

client.setServer(mosquittoServer, 1883);

Вивід інформації про пристрій та його розробника:

```
const char iot_device_banner[] = "IoT пристрій моніторингу метеопараметрів";
const char author_info[] = "Розробив ст. КН-61М, Олег Кубрак";
lcd.print(iot_device_banner);
  lcd.setCursor(0, 1);
  lcd.print(author_info);
  for(int posCount = 0; posCount < 23; posCount++)
  {
    lcd.scrollDisplayLeft(); //builtin command to scroll left the text
    delay(300); // затримка на 300 мс
  }
  lcd.clear();
```

Далі розпочинається цикл вимірювань параметрів мікроклімату, передачі їх значень на MQTT-сервер та виводу інформації на РК-дисплей.

```
void loop() {
```

Зчитування та збереження значень атмосферного тиску, температури і вологості з сенсора BME280 виконується наступними методами об’єкта bme:

```
// зчитуємо температуру, вологість і тиск з сенсора BME280
  temp = bme.readTemperature();
  lcd.setCursor(0, 0);
  lcd.write((byte) 0); lcd.print(' ');
  lcd.print(temp,1);
  lcd.print(char(223));
  lcd.print("C ");
  Serial.print("Температура = ");
  Serial.print(temp);
  Serial.write(176);
  Serial.println("C");

  humidity = bme.readHumidity();
```

```
lcd.write((byte) 1); lcd.print(' ');
lcd.print(humidity, 1);
lcd.print(F("% ")); //lcd.print(F("% RH "));
Serial.print("Вологість = ");
Serial.print(humidity);
Serial.println(" %");

pressure = bme.readPressure() / 100.0F * 0.7500616827;
lcd.setCursor(0, 1);
lcd.write((byte) 2); lcd.print(' ');
lcd.print(pressure, 1); //(int)pressure);
lcd.print(" мм.рт.ст");
lcd.print(" mmHg");
Serial.print("Атмосферний тиск = ");
Serial.print(pressure);
Serial.println(" мм.рт.ст.");
altetude = bme.readAltitude(SEA_LEVEL_PRESSURE_HPA);
Serial.print("Висота = ");
Serial.print(altetude);
Serial.println(" м");
Serial.println();
```

Після збору даних виконується підключення до мережі Wi-Fi та відсилання метеоданих на MQTT-сервер у вигляді повідомлень. MQTT повідомлення публікуються на визначені топіки (теми) за допомогою методу publish, який приймає в якості аргументів топік (тему) і повідомлення.

Підключення до мережі Wi-Fi:

```
status = WiFi.status();
if (status != WL_CONNECTED) {
  while (status != WL_CONNECTED) {
    Serial.print(F("Спроба підключитися до WPA SSID: "));
```

```
      Serial.println(WIFI_AP);
      // підключення до мережі WPA/WPA2
      status = WiFi.begin(WIFI_AP, WIFI_PASSWORD);
      delay(500);
     }
     Serial.println(F("Підключено до AP")); //Connected to AP");
    }
```

Далі виконується спроба підключення до сервера MQTT у циклі до тих пір, поки не буде успішно встановлено з’єднання з сервером:

```
    if (!client.connected() ) {
     reconnect();
    }
    client.loop();
```

Відсилаємо дані кожних 3 секунди:

```
   if ( millis() - lastSend > 3000 ) { // поновлення і відсилка даних через 1 секунду
     sendTempHumData(temp, hum);
     lastSend = millis();
    }

  // {"temp": 27.83, "humidity": 62.05, "pressure": 1013.25}
    /*const int capacity = JSON_OBJECT_SIZE(3);
    StaticJsonDocument<capacity> doc;
    // DynamicJsonDocument doc(capacity);
    doc["temp"] = temp;
    doc["humidity"] = humdidty;
    doc["pressure"] = pressure;
    char jsonBufferSize[512];
    size_t jsonMessageSize = serializeJson(doc, jsonBuffer);
    client.publish(topic, jsonBuffer, jsonMessageSize);*/
```

```
// {"sensor": "bme280", "data": [27.83, 62.05, 1013.25]}
//const int capacity = JSON_ARRAY_SIZE(3) + JSON_OBJECT_SIZE(2);
//StaticJsonDocument<capacity> doc;
/*DynamicJsonDocument doc(capacity); //1024
doc["sensor"] = "bme280";
JsonArray data = doc.createNestedArray("data");
data.add(temp);
data.add(humidity);
data.add(pressure);
serializeJson(doc, Serial);*/

// {"sensorId":"bme280", "data":[{"temp": 27.83}, {"humidity": 62.05}, {"pressure": 1013.25}]}
const int capacity = JSON_ARRAY_SIZE(3) + 3*JSON_OBJECT_SIZE(1) + JSON_OBJECT_SIZE(2);
StaticJsonDocument<capacity> doc;
// DynamicJsonDocument doc(capacity);
doc["sensorId"] = bme.sensorID();
JsonArray data = doc.createNestedArray("data");
JsonObject temp_obj = data.createNestedObject();
temp_obj["temp"] = temp;
JsonObject humidity_obj = data.createNestedObject();
humidity_obj["humidity"] = round(humidity*100)/100.0;
JsonObject pressure_obj = data.createNestedObject();
pressure_obj["pressure"] = round(pressure*10)/10.0;
jsonMessageSize = serializeJson(doc, jsonBuffer);

Serial.println("Відсилаємо повідомлення на MQTT топік..."); //"Sending message to MQTT topic..");
serializeJson(doc, Serial);
```

```
    Serial.println();

    // підключення до Wi-Fi AP і передача даних на MQTT сервер
    /*status = WiFi.status();
    if (status != WL_CONNECTED) {
      while (status != WL_CONNECTED) {
        Serial.print(F("Спроба підключитися до WPA SSID: ")); //Attempting to connect to WPA SSID: ");
        Serial.println(WIFI_SSID);
        // підключення до мережі WPA/WPA2
        status = WiFi.begin(WIFI_SSID, WIFI_PASSWD);
        delay(500);
      }
      Serial.println(F("Підключено до AP")); //Connected to AP");
    }

    if (!client.connected() ) {
      reconnect();
    }

    if ( millis() - lastSend >= 1000 ) { // поновлення і відсилка даних через 1 секунду
        if (client.publish(topic, jsonBuffer, jsonMessageSize) == true) {
          Serial.println("Повідомлення відіслано успішно"); //Success sending message");
        } else {
          Serial.println("Помилка відсилання повідомлення"); //"Error sending message");
        }
        Serial.println();
```

```
      //client.publish(topic, jsonBuffer, jsonMessageSize);
      lastSend = millis();
    }
   client.loop();*/
```

Визначаємо функцію, яка буде викликатися при виникненні зовнішнього переривання INT2. Зовнішнє переривання генерується при натисненні кнопки MENU_BTN підключеної до піна 19 (INT2). Переривання генерується кожний раз при зміні стану вивода (піна) – режим CHANGE.

В головному циклі обробляється лише натиснення кнопки MENU_BTN. При її натисненні відбувається перехід в головне меню, в якому користувач може провести налаштування IoT – пристрою моніторингу метеопараметрів:

```
if (mainMenuInvoke)
  mainMenu();

void ISR_BtnPressed() {
  /*buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);.*/
  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (digitalRead(MENU_BTN_PIN) == HIGH) //isBtnPressed(MENU_BTN_PIN))
  {
    detachInterrupt(digitalPinToInterrupt(interruptPin));
    mainMenuInvoke = true;
  }
}
```

## 4.2. Розроблення програмної бібліотеки для роботи з цифровим сенсором атмосферного тиску, температури і вологості BME280

Для роботи з цифровим сенсором атмосферного тиску, температури і вологості BME280 розроблено на мові C бібліотеку bme280. Вона містить

наступні константи, змінні, структури даних і клас Adafruit_BME280 з методами для роботи з сенсором:

```
/*!
* @file Adafruit_BME280.h
*
*/

#ifndef __BME280_H__
#define __BME280_H__

#include "Arduino.h"

#include <Adafruit_Sensor.h>
#include <SPI.h>
#include <Wire.h>

/*!
 *  I2C адрес за замовчуванням
 */
#define BME280_ADDRESS (0x77)           // первинний I2C адрес
#define BME280_ADDRESS_ALTERNATE (0x76) // альтернативний адрес

/*!
 *  Адреси регістрів
 */
enum {
  BME280_REGISTER_DIG_T1 = 0x88,
  BME280_REGISTER_DIG_T2 = 0x8A,
  BME280_REGISTER_DIG_T3 = 0x8C,

  BME280_REGISTER_DIG_P1 = 0x8E,
  BME280_REGISTER_DIG_P2 = 0x90,
  BME280_REGISTER_DIG_P3 = 0x92,
  BME280_REGISTER_DIG_P4 = 0x94,
  BME280_REGISTER_DIG_P5 = 0x96,
  BME280_REGISTER_DIG_P6 = 0x98,
  BME280_REGISTER_DIG_P7 = 0x9A,
  BME280_REGISTER_DIG_P8 = 0x9C,
  BME280_REGISTER_DIG_P9 = 0x9E,

  BME280_REGISTER_DIG_H1 = 0xA1,
  BME280_REGISTER_DIG_H2 = 0xE1,
  BME280_REGISTER_DIG_H3 = 0xE3,
  BME280_REGISTER_DIG_H4 = 0xE4,
  BME280_REGISTER_DIG_H5 = 0xE5,
  BME280_REGISTER_DIG_H6 = 0xE7,

  BME280_REGISTER_CHIPID = 0xD0,
  BME280_REGISTER_VERSION = 0xD1,
  BME280_REGISTER_SOFTRESET = 0xE0,
  BME280_REGISTER_CAL26 = 0xE1, // R калібрування збережене в 0xE1-0xF0

  BME280_REGISTER_CONTROLHUMID = 0xF2,
  BME280_REGISTER_STATUS = 0XF3,
  BME280_REGISTER_CONTROL = 0xF4,
  BME280_REGISTER_CONFIG = 0xF5,
  BME280_REGISTER_PRESSUREDATA = 0xF7,
  BME280_REGISTER_TEMPDATA = 0xFA,
  BME280_REGISTER_HUMIDDATA = 0xFD
```

```
};

/**************************************************************************/
/*!
    Дані калібрування
*/
/**************************************************************************/
typedef struct {
  uint16_t dig_T1; // компенсаційне значення температури
  int16_t dig_T2;  // компенсаційне значення температури
  int16_t dig_T3;  // компенсаційне значення температури

  uint16_t dig_P1; // компенсаційне значення тиску
  int16_t dig_P2;  // компенсаційне значення тиску
  int16_t dig_P3;  / компенсаційне значення тиску
  int16_t dig_P4;  // компенсаційне значення тиску
  int16_t dig_P5;  // компенсаційне значення тиску
  int16_t dig_P6;  // компенсаційне значення тиску
  int16_t dig_P7;  // компенсаційне значення тиску
  int16_t dig_P8;  // компенсаційне значення тиску
  int16_t dig_P9;  // компенсаційне значення тиску

  uint8_t dig_H1; // компенсаційне значення вологості
  int16_t dig_H2; // компенсаційне значення вологості
  uint8_t dig_H3; // компенсаційне значення вологості
  int16_t dig_H4; // компенсаційне значення вологості
  int16_t dig_H5; // компенсаційне значення вологості
  int8_t dig_H6;  // компенсаційне значення вологості
} bme280_calib_data;
/*=========================================================================*/

/**************************************************************************/
/*!
    Клас, який зберігає стан та функції для взаємодії з чіпом BME280
*/
/**************************************************************************/
class Adafruit_BME280 {
public:
  /**************************************************************************/
  /*!
      Частоти вибірки (замірів)
  */
  /**************************************************************************/
  enum sensor_sampling {
    SAMPLING_NONE = 0b000,
    SAMPLING_X1 = 0b001,
    SAMPLING_X2 = 0b010,
    SAMPLING_X4 = 0b011,
    SAMPLING_X8 = 0b100,
    SAMPLING_X16 = 0b101
  };
  /**************************************************************************/
  /*!
      Режими роботи сенсора
  */
  /**************************************************************************/
  enum sensor_mode {
    MODE_SLEEP = 0b00,
    MODE_FORCED = 0b01,
    MODE_NORMAL = 0b11
  };
```

```
  /*************************************************************************/
  /*!
      Значення фільтру
  */
  /*************************************************************************/
  enum sensor_filter {
    FILTER_OFF = 0b000,
    FILTER_X2 = 0b001,
    FILTER_X4 = 0b010,
    FILTER_X8 = 0b011,
    FILTER_X16 = 0b100
  };
  /*************************************************************************/
  /*!
      Тривалість очікування в мсекс.
  */
  /*************************************************************************/
  enum standby_duration {
    STANDBY_MS_0_5 = 0b000,
    STANDBY_MS_10 = 0b110,
    STANDBY_MS_20 = 0b111,
    STANDBY_MS_62_5 = 0b001,
    STANDBY_MS_125 = 0b010,
    STANDBY_MS_250 = 0b011,
    STANDBY_MS_500 = 0b100,
    STANDBY_MS_1000 = 0b101
  };
  // Конструктори
  Adafruit_BME280();
  Adafruit_BME280(int8_t cspin, SPIClass *theSPI = &SPI);
      Adafruit_BME280(int8_t cspin, int8_t mosipin, int8_t misopin,
                      int8_t sckpin);

  bool begin();
  bool begin(TwoWire *theWire);
  bool begin(uint8_t addr);
  bool begin(uint8_t addr, TwoWire *theWire);
  bool init();

  void setSampling(sensor_mode mode = MODE_NORMAL,
                   sensor_sampling tempSampling = SAMPLING_X16,
                   sensor_sampling pressSampling = SAMPLING_X16,
                   sensor_sampling humSampling = SAMPLING_X16,
                   sensor_filter filter = FILTER_OFF,
                   standby_duration duration = STANDBY_MS_0_5);

  void takeForcedMeasurement();
  float readTemperature(void);
  float readPressure(void);
  float readHumidity(void);

  float readAltitude(float seaLevel);
  float seaLevelForAltitude(float altitude, float pressure);
  uint32_t sensorID(void);

protected:
  TwoWire *_wire; //! вказівник на об’єкт TwoWire
  SPIClass *_spi; //! вказівник на об’єкт SPI
  void readCoefficients(void);
  bool isReadingCalibration(void);
  uint8_t spixfer(uint8_t x);
```

```
  void write8(byte reg, byte value);
  uint8_t read8(byte reg);
  uint16_t read16(byte reg);
  uint32_t read24(byte reg);
  int16_t readS16(byte reg);
  uint16_t read16_LE(byte reg); // little endian
  int16_t readS16_LE(byte reg); // little endian

  uint8_t _i2caddr;  // I2C адрес для інтерфейсу TwoWire
  int32_t _sensorID; // ID сенсора BME
  int32_t t_fine; // температура з високою роздільною здатністю, збережена як атрибут
                 // це значення використовується для компенсації зчитаних значень
температури
                  // вологість і тиск
  int8_t _cs;   // для інтерфейсу SPI
  int8_t _mosi; // для інтерфейсу SPI
  int8_t _miso; // для інтерфейсу SPI
  int8_t _sck;  // для інтерфейсу SPI

 bme280_calib_data _bme280_calib; // збережені дані калібрування
  /**************************************************************************/
  /*!
      Конфігураційний регістр
  */
  /**************************************************************************/
  struct config {
    // неактивна тривалість (час очікування) в нормальному режимі
    // 000 = 0.5 ms
    // 001 = 62.5 ms
    // 010 = 125 ms
    // 011 = 250 ms
    // 100 = 500 ms
    // 101 = 1000 ms
    // 110 = 10 ms
    // 111 = 20 ms
    unsigned int t_sb : 3; // неактивна тривалість (час очікування) в нормальному режимі
    // налаштування фільтра
    // 000 = filter off
    // 001 = 2x filter
    // 010 = 4x фільтр
    // 011 = 8x фільтр
    // 100 і вище = 16x фільтр
    unsigned int filter : 3; // налаштування фільтра
    // unused - don't set
    unsigned int none : 1;     // не використовується – не встановлювати
    unsigned int spi3w_en : 1; // не використовується – не встановлювати
    /// повертає combined конфігураційний регістр
    unsigned int get() { return (t_sb << 5) | (filter << 2) | spi3w_en; }
  };
  config _configReg; // об’єкт конфігураційного регістра
  /**************************************************************************/
  /*!
      Регістр ctrl_meas
  */
  /**************************************************************************/
  struct ctrl_meas {
    // вибірка температури з запасом по частоті (дискретизація температури з вищою
частотою)
    // 000 = пропущено
    // 001 = x1
    // 010 = x2
```

```
    // 011 = x4
    // 100 = x8
    // 101 і вище = x16
    unsigned int osrs_t : 3; // дискретизація температури з вищою частотою

    // вибірка тиску з запасом по частоті (дискретизація тиску з вищою частотою)
    // 000 = пропущено
    // 001 = x1
    // 010 = x2
    // 011 = x4
    // 100 = x8
    // 101 і вище = x16
    unsigned int osrs_p : 3; // дискретизація тиску з вищою частотою
    // режими пристрою
    // 00       = сон
    // 01 або 10 = forced
    // 11       = нормальний (робочий)
    unsigned int mode : 2; / режим роботи пристрою

    // повертає combined ctrl регістр
    unsigned int get() { return (osrs_t << 5) | (osrs_p << 2) | mode; }
  };
  ctrl_meas _measReg; // об’єкт регістра вимірювань
  /*************************************************************************/
  /*!
      Регістр ctrl_hum
  */
  /*************************************************************************/
  struct ctrl_hum {
    // не використовується – не встановлювати
    unsigned int none : 5;
    // вибірка тиску з запасом по частоті (дискретизація тиску з вищою частотою)
    // 000 = skipped
    // 001 = x1
    // 010 = x2
    // 011 = x4
    // 100 = x8
    // 101 і вище = x16
    unsigned int osrs_h : 3; // дискретизація тиску з вищою частотою
    // повертає combined ctrl hum регістр
    unsigned int get() { return (osrs_h); }
  };
  ctrl_hum _humReg; // об’єкт регістра вологості hum
};

#endif
```

## 4.3. Розроблення програмної бібліотеки для роботи з Wi-Fi модулем ESP8266

Для роботи з Wi-Fi модулем ESP8266 ESP-01S розроблено бібліотеку, що включає наступні функції:

```
#ifndef WiFiEsp_h
#define WiFiEsp_h

#include <Arduino.h>
#include <Stream.h>
```

```
#include <IPAddress.h>
#include <inttypes.h>

#include "WiFiEspClient.h"
#include "WiFiEspServer.h"
#include "utility/EspDrv.h"
#include "utility/RingBuffer.h"
#include "utility/debug.h"

class WiFiEspClass
{

  public:
     static int16_t _state[MAX_SOCK_NUM];
     static uint16_t _server_port[MAX_SOCK_NUM];
     WiFiEspClass();
     /* Ініціалізувати ESP модуль.
      * param espSerial: послідовний інтерфейс (HW або SW) використовується для
      * комунікації з модулем ESP
     */
     static void init(Stream* espSerial);
      /* Прочитати версію прошивки (firmware version) */
      static char* firmwareVersion();

      /* Стартує Wifi підключення з ідентифікацією
      * найбільш захищений режим, що підтримується, буде автоматично вибрано
      * param ssid: Вказівник на рядок SSID.
      * param passphrase: Пароль. Дозволений (допустимий) набір символів у паролі має
      * бути між ASCII 32-126 (в десятковій системі числення).
     */
     int begin(const char* ssid, const char* passphrase);

     /* Змінює налаштування Ip конфігурації відключивши клієнт DHCP
      * param local_ip: статична ip конфігурація
     */
     void config(IPAddress local_ip);

     /* Відключення від мережі
      * return: значення wl_status_t enum
     */
     int disconnect(void);

     /* Прочитати MAC адресу інтерфейсу.
      * return: вказівник на uint8_t array довжиною WL_MAC_ADDR_LENGTH
     */
     uint8_t* macAddress(uint8_t* mac);

    /* Отримати інтерфейсний IP адрес.
     * return: значення Ip адресу
    */
    IPAddress localIP();

    /* Отримати адресної маски підмережі інтерфейсу.
     * return: значення адресної маски підмережі subnet mask address value
    */
    IPAddress subnetMask();

    /* Отримати ip адресу шлюзу.
     * return: значення ip адресу шлюза
    */
    IPAddress gatewayIP();
```

```
/* Повертає поточний SSID зв’язаний з мережею
 * return: рядок ssid
*/
char* SSID();

/* Повертає поточний BSSID зв’язаний з мережею.
 * Це MAC адрес точки доступу (Access Point (AP))
 * return: вказівник на uint8_t масив довжиною WL_MAC_ADDR_LENGTH
*/
uint8_t* BSSID(uint8_t* bssid);

/* Повертає поточний RSSI /Отримана потужність сигналу
 * (Received Signal Strength) в dBm) зв’язного з мережею
 * return: знакове значення
*/
int32_t RSSI();

/* Повертарє статус (стан) підключення.
 * return: одне з значень визначених в wl_status_t
*/
uint8_t status();

/* Сканує доступні мережі WiFi
 * return: Число виявлених мереж
*/
int8_t scanNetworks();

/* Повертає SSID виявлені при скануванні мереж.
 * param networkItem: визначає з якої мережі хочемо отримати інформацію
 * return: рядок ssid визначеної мережевої точки зі списку відсканованих мереж.
*/
char* SSID(uint8_t networkItem);

/* Повертає тип шифрування мереж виявлених при скануванні scanNetworks
 * param networkItem: визначає з якої мережі хочемо отримати інформацію
 * return: тип шифрування (enum wl_enc_type)
 */
uint8_t  encryptionType(uint8_t networkItem);

/* Повертає RSSI мереж виявлених при скануванні scanNetworks
 * param networkItem: визначає з якої мережевої точки хочемо отримати інформацію
 * return: значення RSSI визначеної точки (одиниці) зі списку відсканованих мереж
*/
int32_t RSSI(uint8_t networkItem);

/* Стартує точка доступу ESP.
 * param ssid: Вказівник на рядок SSID.
 * param channel: WiFi канал (1-14)
 * param pwd: Пароль має містити допустимі ASCII символи 32-126
 * (в десятковій системі).
 * param enc: тип шифрування (enum wl_enc_type)
 * param apOnly: встановлюємо false якщо хочемо запустити AP
 * і режими станції одночасно
*/
int beginAP(const char* ssid, uint8_t channel, const char* pwd, uint8_t enc, bool
apOnly=true);

/* Старт ESP точки доступу з відкритим доступом */
int beginAP(const char* ssid);
int beginAP(const char* ssid, uint8_t channel);
```

```
    /* Зміна IP адресу AP
     * param ip: Статична конфігурація ip
    */
    void configAP(IPAddress ip);

    /* Перезавантаження (перезапуск) модуля ESP */
    void reset();

    /* Ping хоста */
    bool ping(const char *host);

    friend class WiFiEspClient;
    friend class WiFiEspServer;
    friend class WiFiEspUDP;

  private:
    static uint8_t getFreeSocket();
    static void allocateSocket(uint8_t sock);
    static void releaseSocket(uint8_t sock);

    static uint8_t espMode;
};

extern WiFiEspClass WiFi;

#endif
```

## 4.4. Програмна реалізація головного алгоритму роботи IoT – системи моніторингу метеопараметрів

Розроблено програмну реалізацію головного алгоритму, що дозволяє збирати, обробляти метеодані та передавати їх по протоколу MQTT на локальний сервер Mosquitto або віддалену IoT – платформу ThingsBoard:

```
void loop() {
  // зчитуємо температуру, вологість і тиск з сенсора BME280
  temp = bme.readTemperature();
  lcd.home(); //setCursor(0, 0);
  lcd.write((byte) 0); lcd.print(' ');
  lcd.print(temp,1);
  lcd.print(char(223));
  lcd.print("C ");
  Serial.print("Температура = ");
  Serial.write(176);
  Serial.println("C");

  humidity = bme.readHumidity();
  lcd.write((byte) 1); lcd.print(' ');
  lcd.print(humidity, 1);
```

```
lcd.print(F("% ")); //lcd.print(F("% RH "));
Serial.print("Вологість = ");
Serial.print(humidity);
Serial.println(" %");

pressure = bme.readPressure() / 100.0F * 0.7500616827;
lcd.setCursor(0, 1);
lcd.write((byte) 2); lcd.print(' ');
lcd.print(pressure, 1); //(int)pressure);
lcd.print(" mmHg");
Serial.print("Атмосферний тиск = ");
Serial.print(pressure);
Serial.println(" мм.рт.ст.");
//Serial.println(" mmHg");

altetude = bme.readAltitude(SEA_LEVEL_PRESSURE_HPA);
//lcd.print(altetude);
//lcd.print(" m");
Serial.print("Висота = ");
Serial.print(altetude);
Serial.println(" м");
Serial.println();

// {"temp": 27.83, "humidity": 62.05, "pressure": 1013.25}
/*const int capacity = JSON_OBJECT_SIZE(3);
StaticJsonDocument<capacity> doc;
// DynamicJsonDocument doc(capacity);
doc["temp"] = temp;
doc["humidity"] = humdidty;
doc["pressure"] = pressure;
char jsonBufferSize[512];
size_t jsonMessageSize = serializeJson(doc, jsonBuffer);
client.publish(topic, jsonBuffer, jsonMessageSize);*/

// {"sensor": "bme280", "data": [27.83, 62.05, 1013.25]}
//const int capacity = JSON_ARRAY_SIZE(3) + JSON_OBJECT_SIZE(2);
//StaticJsonDocument<capacity> doc;
/*DynamicJsonDocument doc(capacity); //1024
doc["sensor"] = "bme280";
JsonArray data = doc.createNestedArray("data");
data.add(temp);
```

```
  data.add(humidity);
  data.add(pressure);
  serializeJson(doc, Serial);*/

  //  {"sensorId":"bme280",  "data":[{"temp":  27.83},  {"humidity":  62.05},  {"pressure":
1013.25}]}
  const int capacity = JSON_ARRAY_SIZE(3) + 3*JSON_OBJECT_SIZE(1) + JSON_OBJECT_SIZE(2);
  StaticJsonDocument<capacity> doc;
  // DynamicJsonDocument doc(capacity);
  doc["sensorId"] = bme.sensorID();
  JsonArray data = doc.createNestedArray("data");
  JsonObject temp_obj = data.createNestedObject();
  temp_obj["temp"] = temp;
  JsonObject humidity_obj = data.createNestedObject();
  humidity_obj["humidity"] = round(humidity*100)/100.0;
  JsonObject pressure_obj = data.createNestedObject();
  pressure_obj["pressure"] = round(pressure*10)/10.0;
  jsonMessageSize = serializeJson(doc, jsonBuffer);

  Serial.println("Відсилаємо повідомлення на MQTT топік..."); //"Sending message to MQTT
topic..");
  serializeJson(doc, Serial);
  Serial.println();

  // підключення до Wi-Fi AP і передача даних на MQTT сервер
  status = WiFi.status();
  if (status != WL_CONNECTED) {
    while (status != WL_CONNECTED) {
      Serial.print(F("Спроба підключитися до WPA SSID: ")); //Attempting to connect to
WPA SSID: ");
      Serial.println(WIFI_SSID);
      // підключення до мережі WPA/WPA2
      status = WiFi.begin(WIFI_SSID, WIFI_PASSWD);
      delay(500);
    }
    Serial.println(F("Підключено до AP")); //Connected to AP");
  }

  if (!client.connected() ) {
    reconnect();
  }
```

```
  if ( millis() - lastSend >= 1000 ) { // поновлення і відсилка даних через 1 секунду
     if (client.publish(topic, jsonBuffer, jsonMessageSize) == true) {
       Serial.println("Повідомлення відіслано успішно"); //Success sending message");
     } else {
       Serial.println("Помилка відсилання повідомлення"); //"Error sending message");
     }
     Serial.println();
     //client.publish(topic, jsonBuffer, jsonMessageSize);
     lastSend = millis();
   }

   client.loop();
  /*if (mainMenuInvoke)
     mainMenu();*/

   delay(delayTime);
}
```

## РОЗДІЛ V. МОДЕЛЮВАННЯ ТА ДОСЛІДЖЕННЯ МАКЕТУ IoT – ПРИСТРОЮ МОНІТОРИНГУ МЕТЕОПАРАМЕТРІВ

Написаний скетч (програму) в Arduino IDE компілюємо і лінкуємо у файл прошивки з розширенням hex для МК ATmega2560 плати Arduino Mega2560. Перед завантаженням прошивки у МК Arduino - плати перевіримо коректність її роботи на створеній моделі пристрою в Proteus. В Proteus можна провести моделювання і відлагодити досить складні пристрої. В таких пристроях може бути подекілька мікроконтролерів одночасно і навіть з різних родин. На рисунках нижче показано моделювання IoT – пристрою моніторингу метеопараметрів в Proteus ISIS.

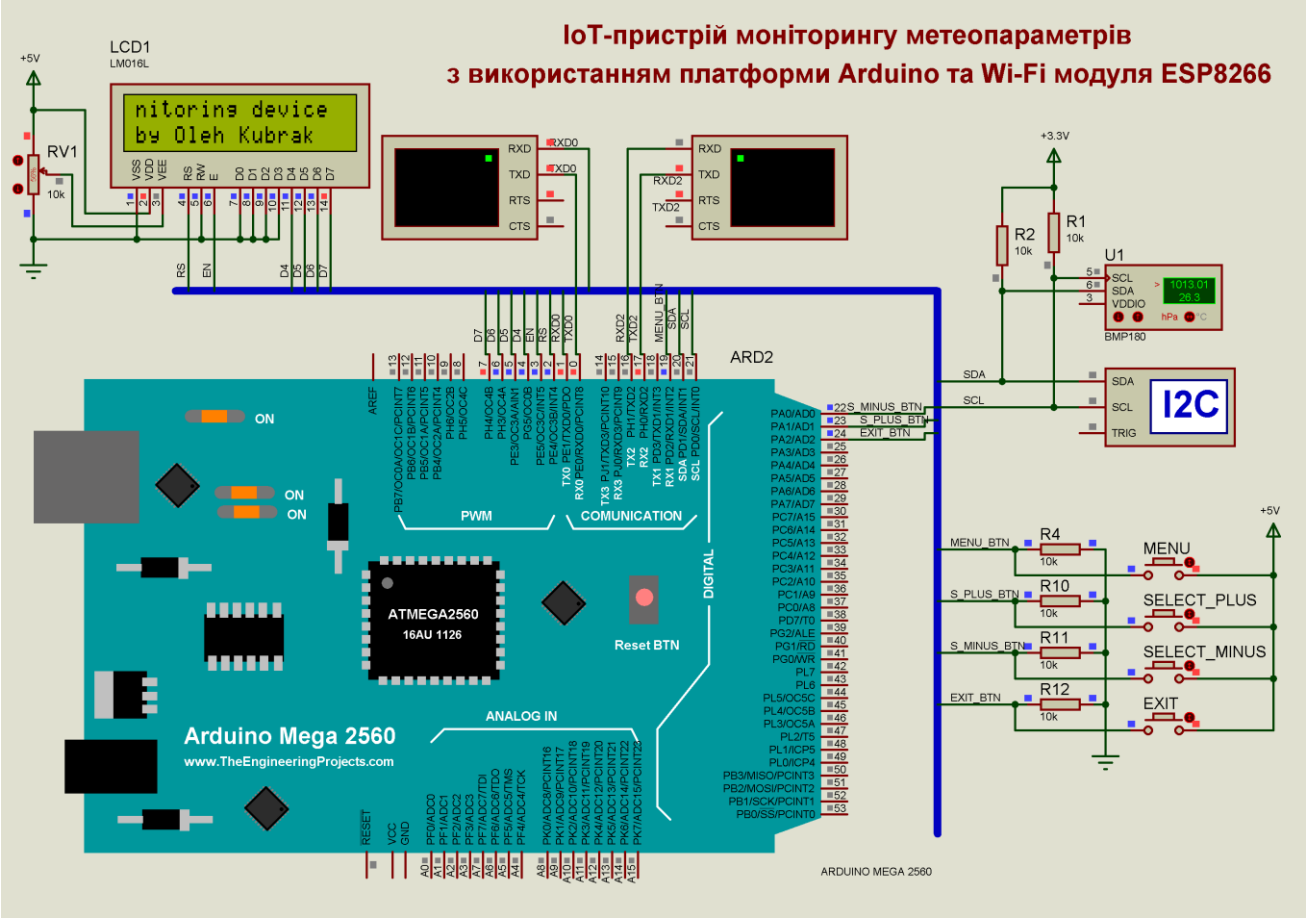



Рис.5.1. Моделювання роботи IoT – пристрою моніторингу метеопараметрів в Proteus ISIS

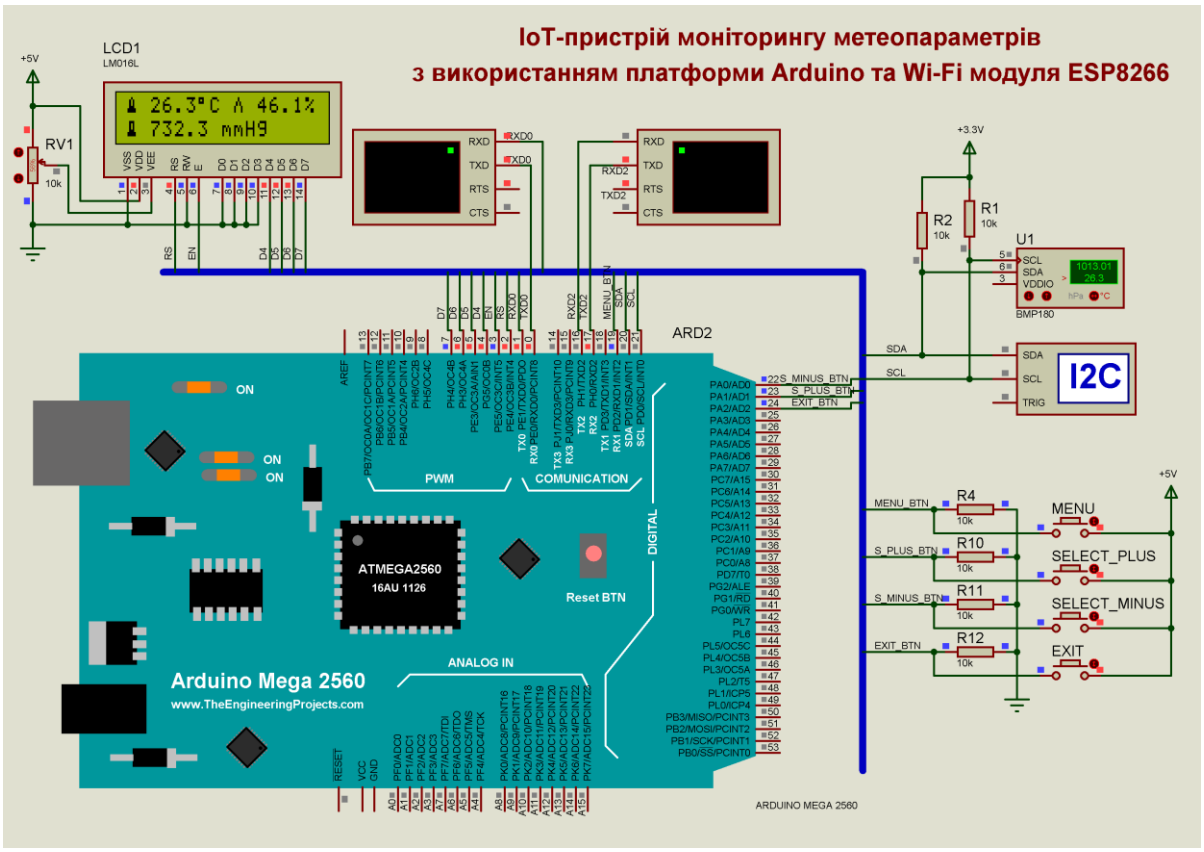



Рис.5.2. Моделювання роботи IoT - пристрою моніторингу метеопараметрів в емуляторі Proteus ISIS

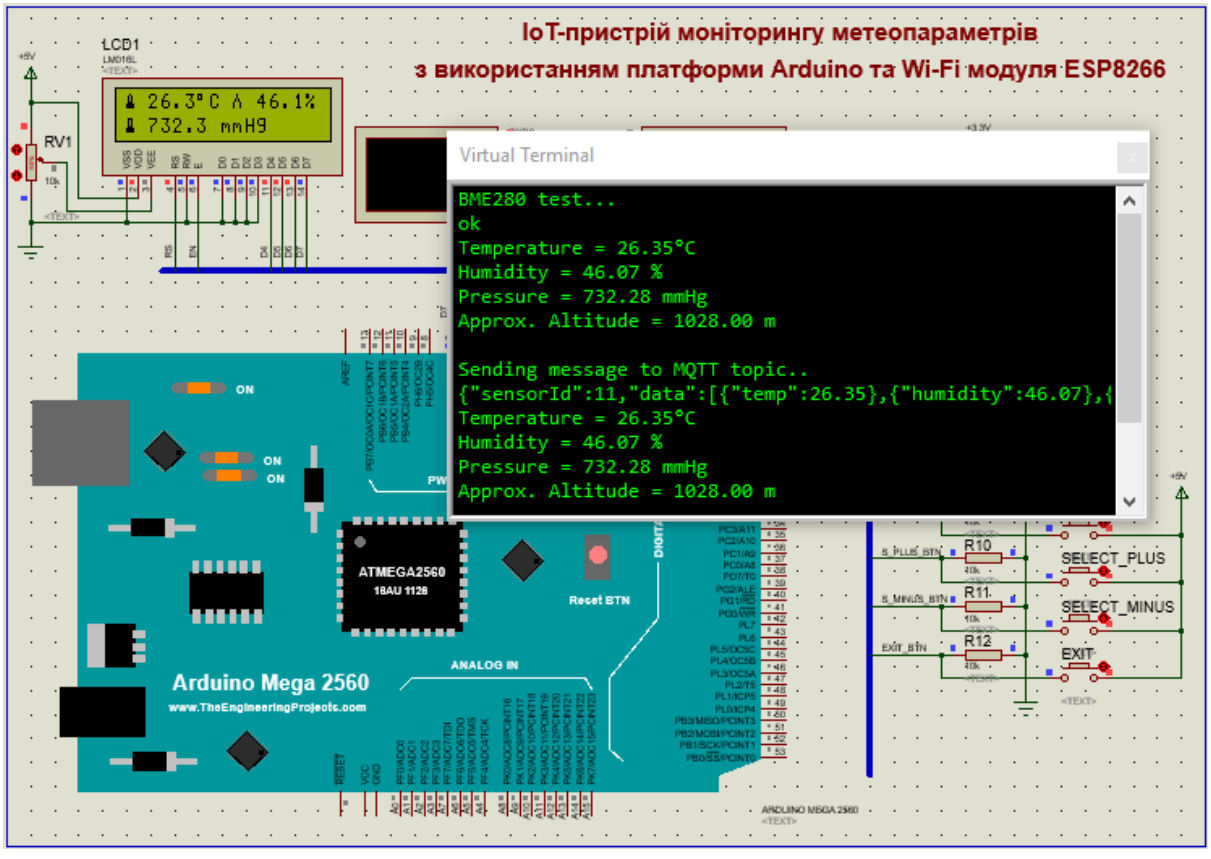
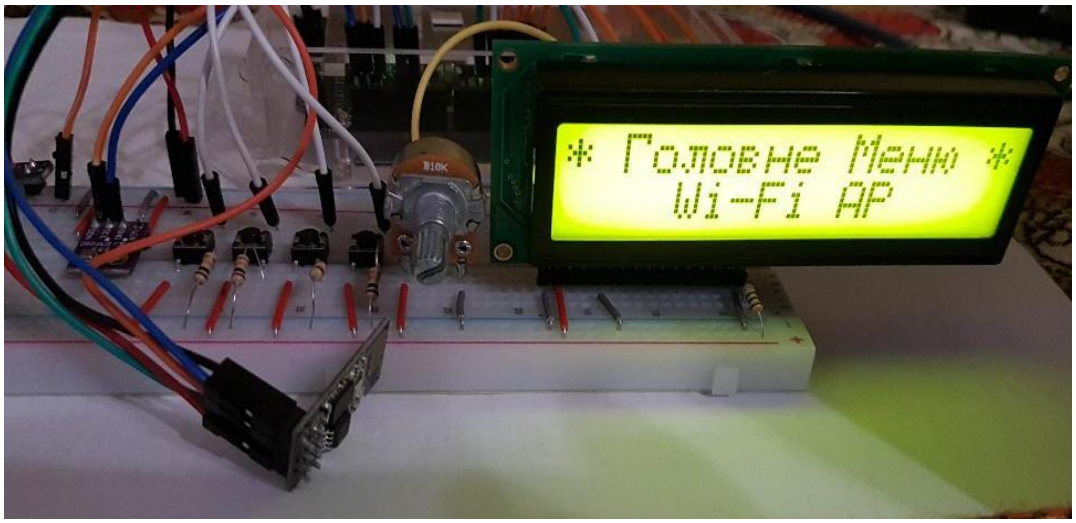
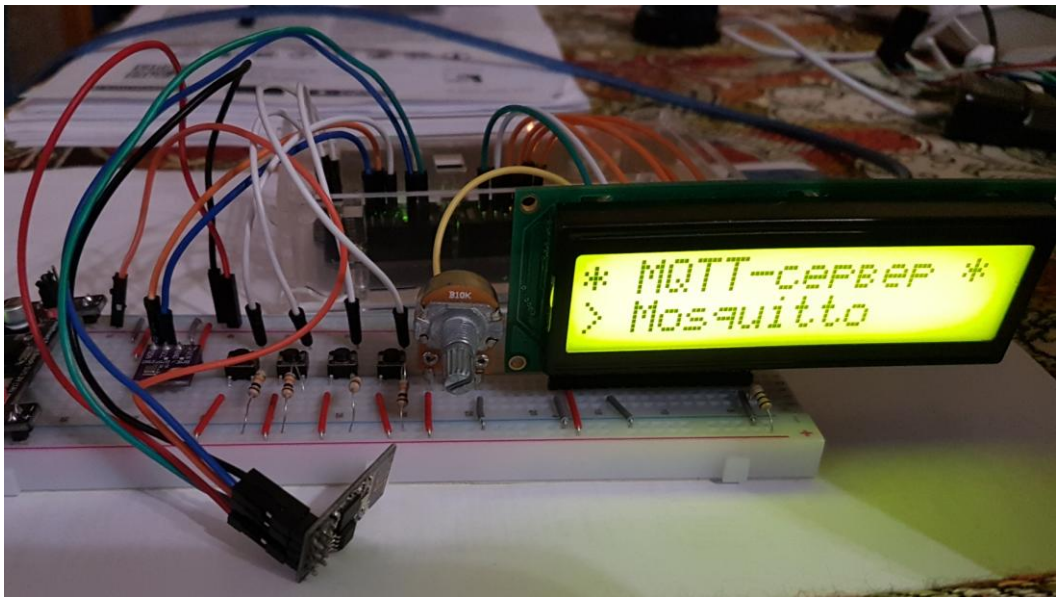


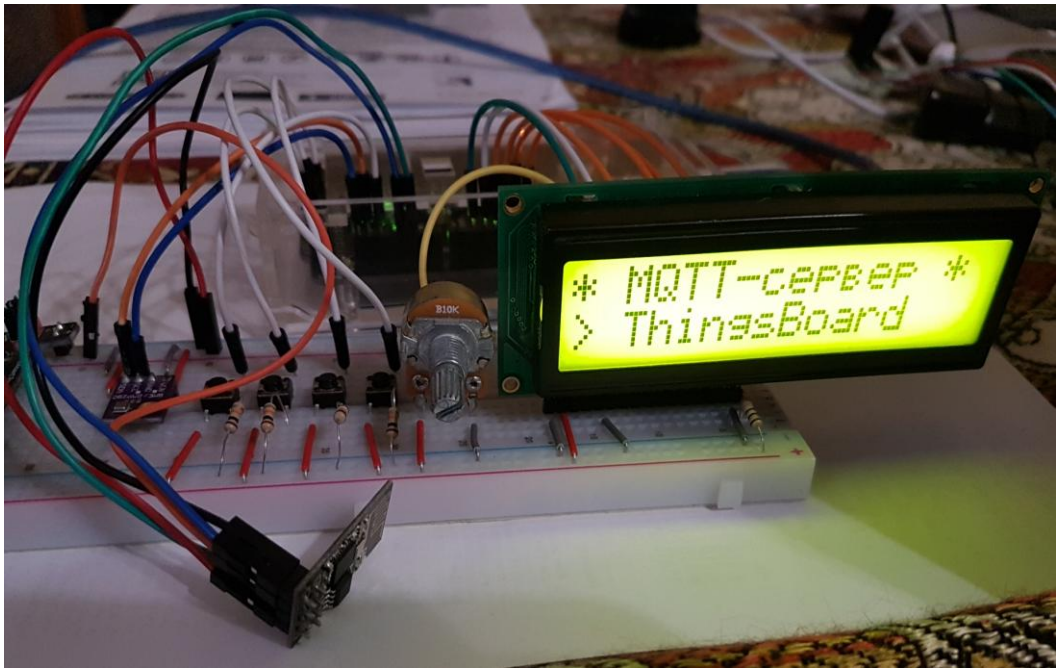

Рис.5.3. Моделювання роботи IoT – пристрою моніторингу метеопараметрів

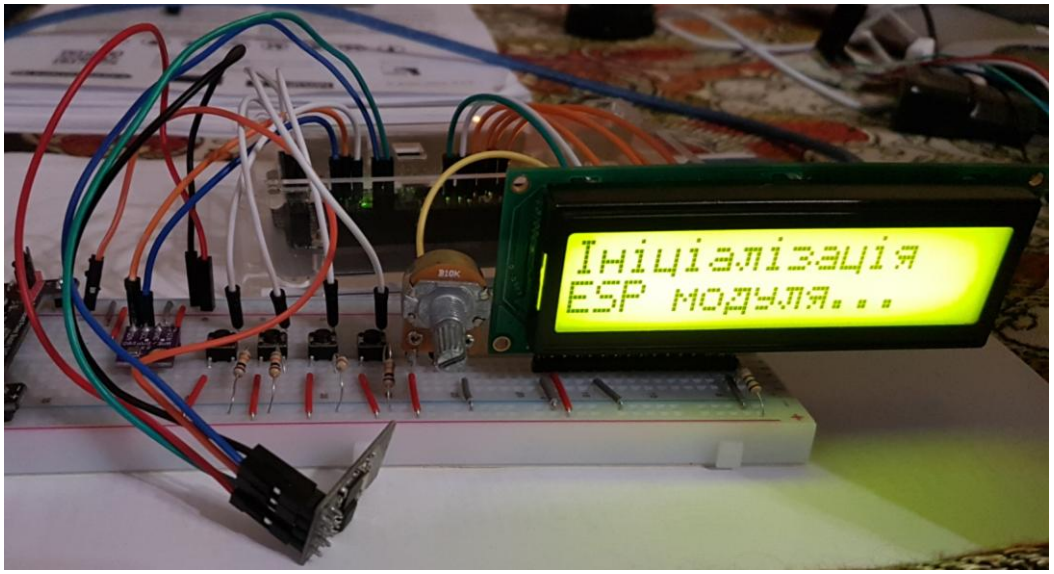
На рисунках нижче показано роботу зібраного макету IoT-пристрою моніторингу метеопараметрів.

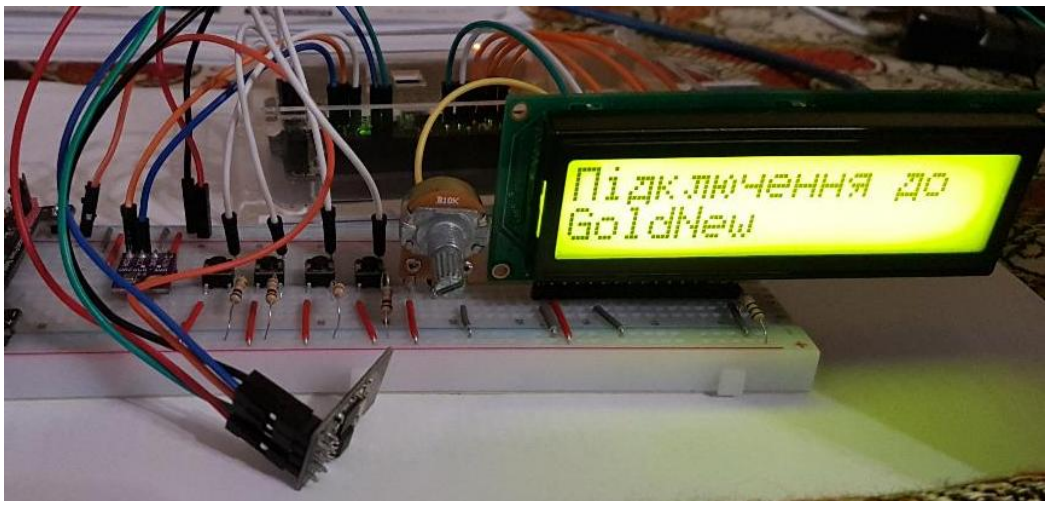


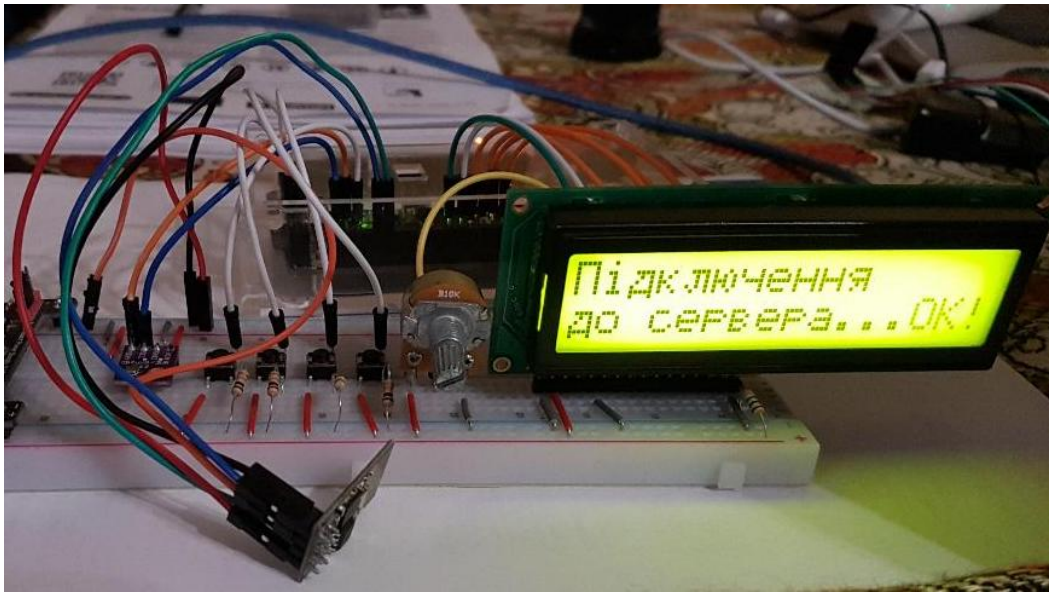


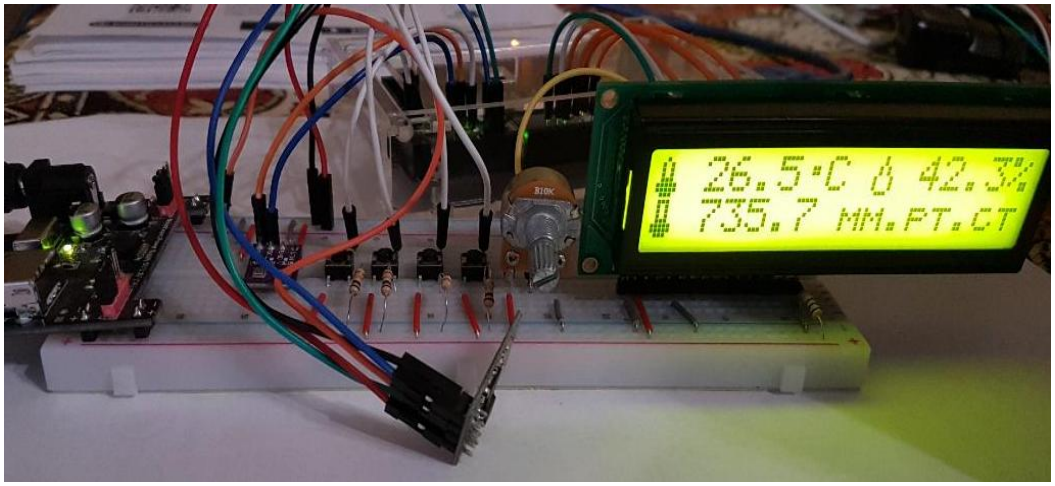

* MQTT-сервер *
> Mosquitto

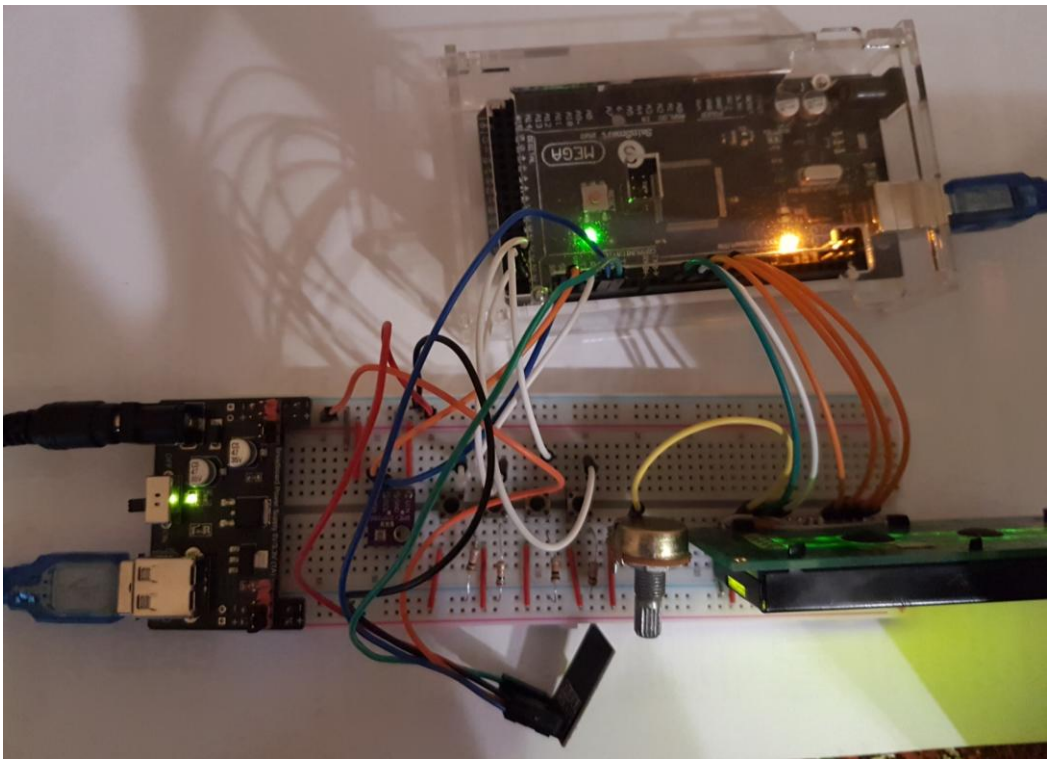

* MQTT-сервер *
> ThingsBoard


Ініціалізація
ESP модуля...


Підключення до
GoldNew


Підключення
до сервера...OK!

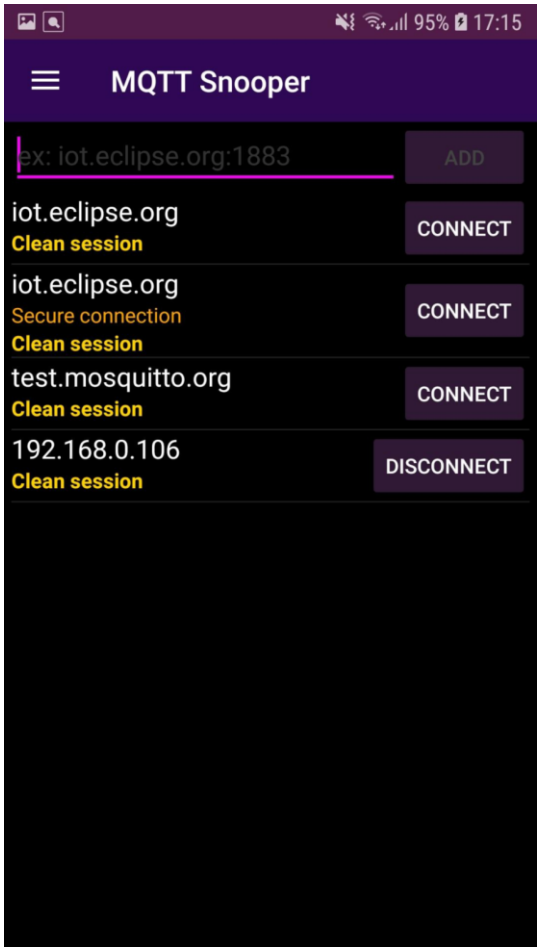

26.5°C 42.3%
735.7 мм.рт.ст

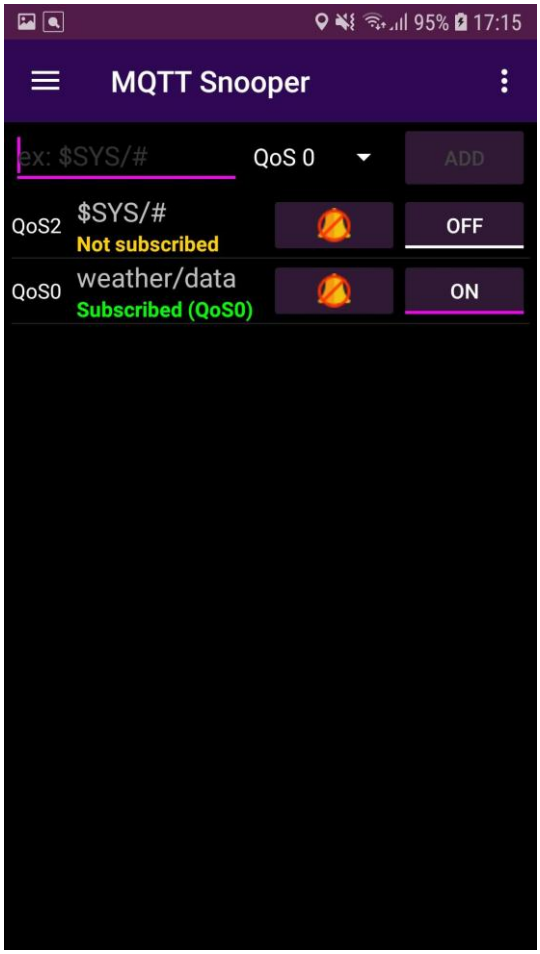


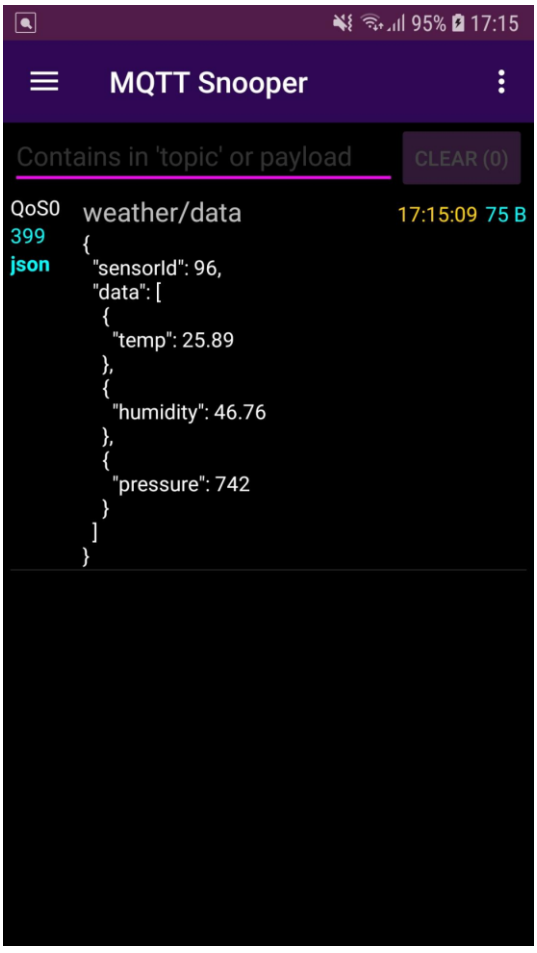
Administrator: Command Prompt

```
home/temperature 25.26
home/humidity 45.85
home/temperature 25.29
home/humidity 45.87
home/temperature 25.29
home/humidity 45.86
home/temperature 25.27
home/humidity 45.96
home/temperature 25.27
home/humidity 46.08

C:\Program Files\mosquitto>mosquitto_sub -h localhost -t home/# -p 1883 -v -C 10
home/temperature 25.23
home/humidity 48.15
home/temperature 25.27
home/humidity 48.21
home/temperature 25.29
home/humidity 48.27
home/temperature 25.29
home/humidity 48.29
home/temperature 25.27
home/humidity 48.26

C:\Program Files\mosquitto>mosquitto_sub -h localhost -t home/# -p 1883 -v -C 10
home/temperature 25.43
home/humidity 47.58
home/temperature 25.45
home/humidity 47.59
home/temperature 25.43
home/humidity 47.43
home/temperature 25.45
home/humidity 47.50
home/temperature 25.43
```

Рис.5.4. Вивід повідомлень на підписані топіки home/temperature і home/humidity, які приходять від MQTT сервера (брокера) Mosquitto


95% 17:15
MQTT Snooper
ex: iot.eclipse.org:1883
ADD
iot.eclipse.org
Clean session
CONNECT
iot.eclipse.org
Secure connection
Clean session
CONNECT
test.mosquitto.org
Clean session
CONNECT
192.168.0.106
Clean session
DISCONNECT


95% 17:15
MQTT Snooper
ex: $SYS/#
QoS 0
ADD
QoS2
$SYS/#
Not subscribed
OFF
QoS0
weather/data
Subscribed (QoS0)
ON


95% 17:15
MQTT Snooper
Contains in 'topic' or payload
CLEAR (0)
QoS0
399
json
weather/data
17:15:09 75 B
{
"sensorId": 96,
"data": [
{
"temp": 25.89
},
{
"humidity": 46.76
},
{
"pressure": 742
}
]
}

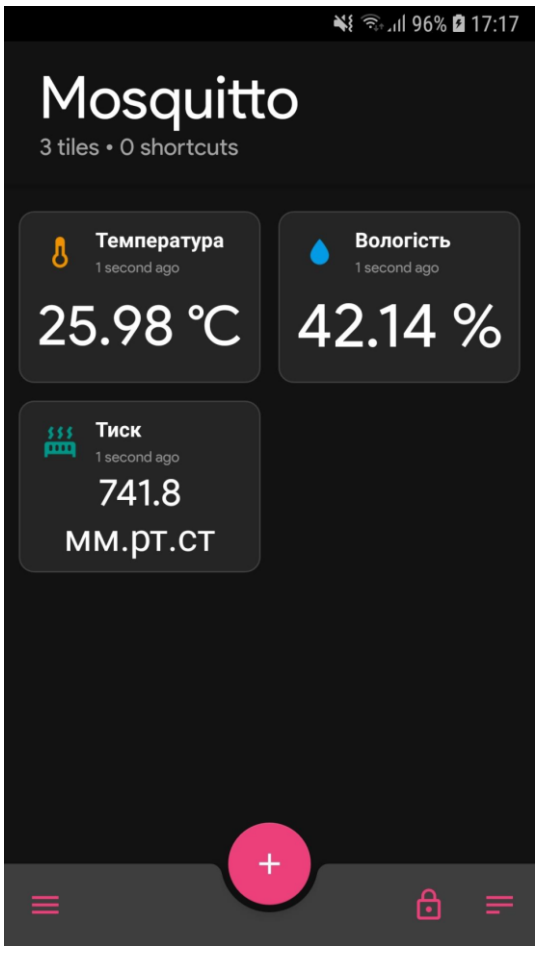
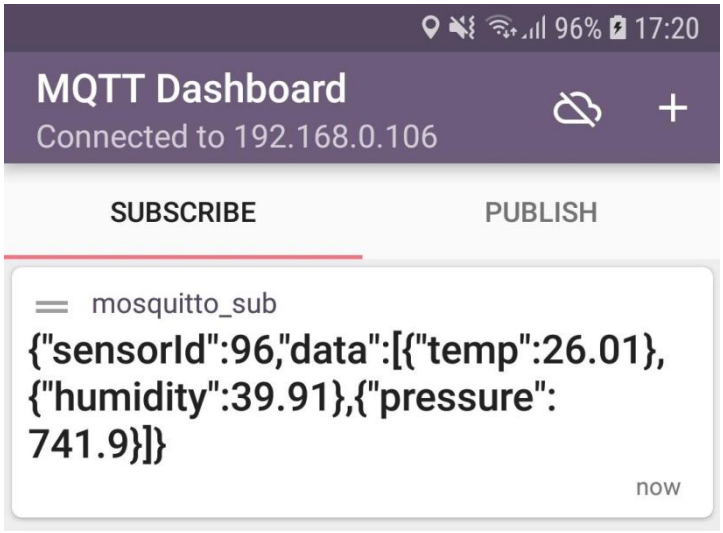
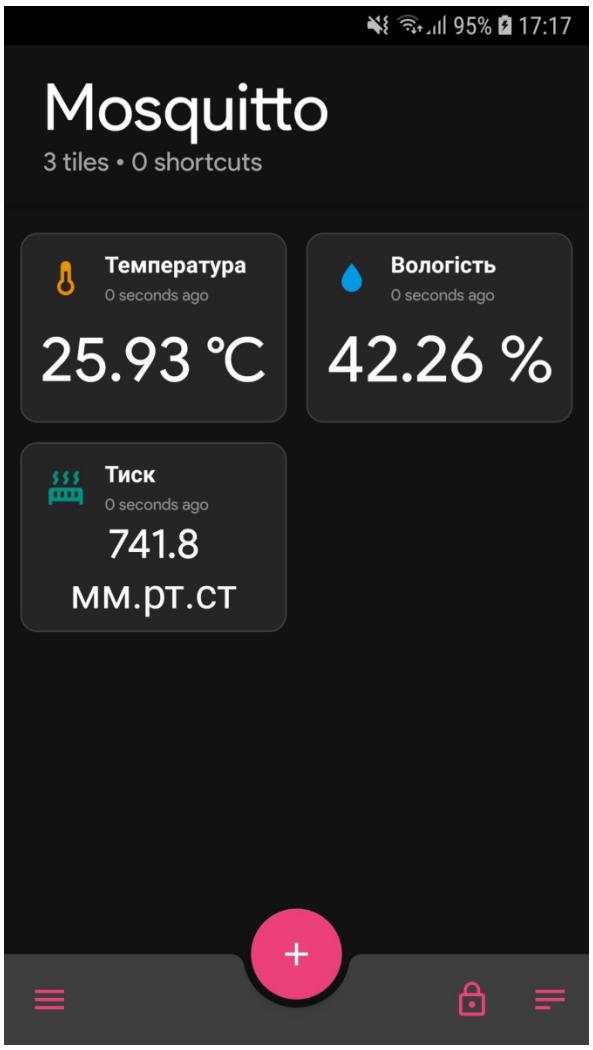

96% 17:17
Mosquitto
3 tiles • 0 shortcuts
Температура
1 second ago
25.98 °C
Вологість
1 second ago
42.14 %
Тиск
1 second ago
741.8
мм.рт.ст

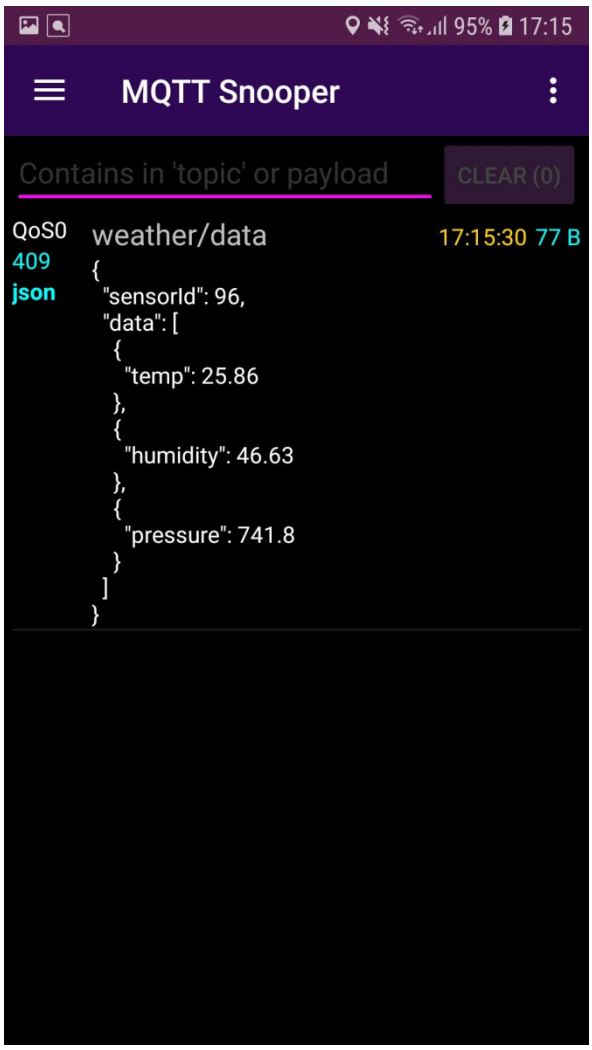

Рис.5.6. Вивід параметрів мікроклімату на Dashboard

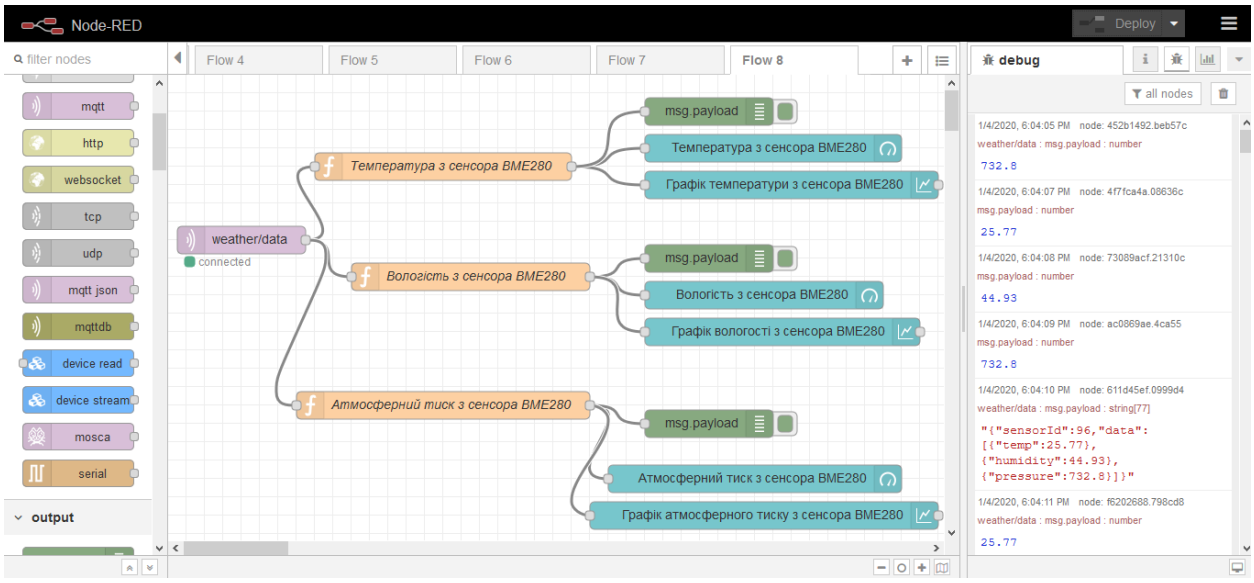

96% 17:17
Dashboard
7 Received Messages
{"sensorId":96,"data":[{"temp":25.98},{"humidity":41.43},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.95},{"humidity":41.54},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.95},{"humidity":41.52},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.97},{"humidity":41.46},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.97},{"humidity":41.51},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.98},{"humidity":41.56},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.97},{"humidity":41.52},{"pressure":741.8}]}
weather/data


96% 17:18
Dashboard
7 Received Messages
{"sensorId":96,"data":[{"temp":25.96},{"humidity":41.28},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.97},{"humidity":41.28},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.98},{"humidity":41.25},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.98},{"humidity":41.35},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.98},{"humidity":41.32},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":25.97},{"humidity":41.51},{"pressure":741.8}]}
weather/data
{"sensorId":96,"data":[{"temp":26},{"humidity":41.44},{"pressure":741.8}]}
weather/data


96% 17:20
MQTT Dashboard
Connected to 192.168.0.106
SUBSCRIBE
PUBLISH
mosquitto_sub
{"sensorId":96,"data":[{"temp":26.01},{"humidity":39.91},{"pressure":741.9}]}
now


95% 17:17
Mosquitto
3 tiles • 0 shortcuts
Температура
0 seconds ago
25.93 °C
Вологість
0 seconds ago
42.26 %
Тиск
0 seconds ago
741.8
мм.рт.ст


95% 17:15
MQTT Snooper
Contains in 'topic' or payload
CLEAR (0)
QoS0
409
json
weather/data
17:15:30 77 B
{
"sensorId": 96,
"data": [
{
"temp": 25.86
},
{
"humidity": 46.63
},
{
"pressure": 741.8
}
]
}


Node-RED
Deploy
filter nodes
Flow 4
Flow 5
Flow 6
Flow 7
Flow 8
debug
all nodes
mqtt
http
websocket
tcp
udp
mqtt json
mqttdb
device read
device stream
mosca
serial
output
weather/data
connected
Температура з сенсора BME280
Вологість з сенсора BME280
Атмосферний тиск з сенсора BME280
msg.payload
Температура з сенсора BME280
Графік температури з сенсора BME280
msg.payload
Вологість з сенсора BME280
Графік вологості з сенсора BME280
msg.payload
Атмосферний тиск з сенсора BME280
Графік атмосферного тиску з сенсора BME280
1/4/2020, 6:04:05 PM node: 452b1492.beb57c
weather/data : msg.payload : number
732.8
1/4/2020, 6:04:07 PM node: 4f7fca4a.08636c
msg.payload : number
25.77
1/4/2020, 6:04:08 PM node: 73089acf.21310c
msg.payload : number
44.93
1/4/2020, 6:04:09 PM node: ac0869ae.4ca55
msg.payload : number
732.8
1/4/2020, 6:04:10 PM node: 611d45ef.0999d4
weather/data : msg.payload : string[77]
"{"sensorId":96,"data":[{"temp":25.77},{"humidity":44.93},{"pressure":732.8}]}"
1/4/2020, 6:04:11 PM node: f6202688.798cd8
weather/data : msg.payload : number
25.77

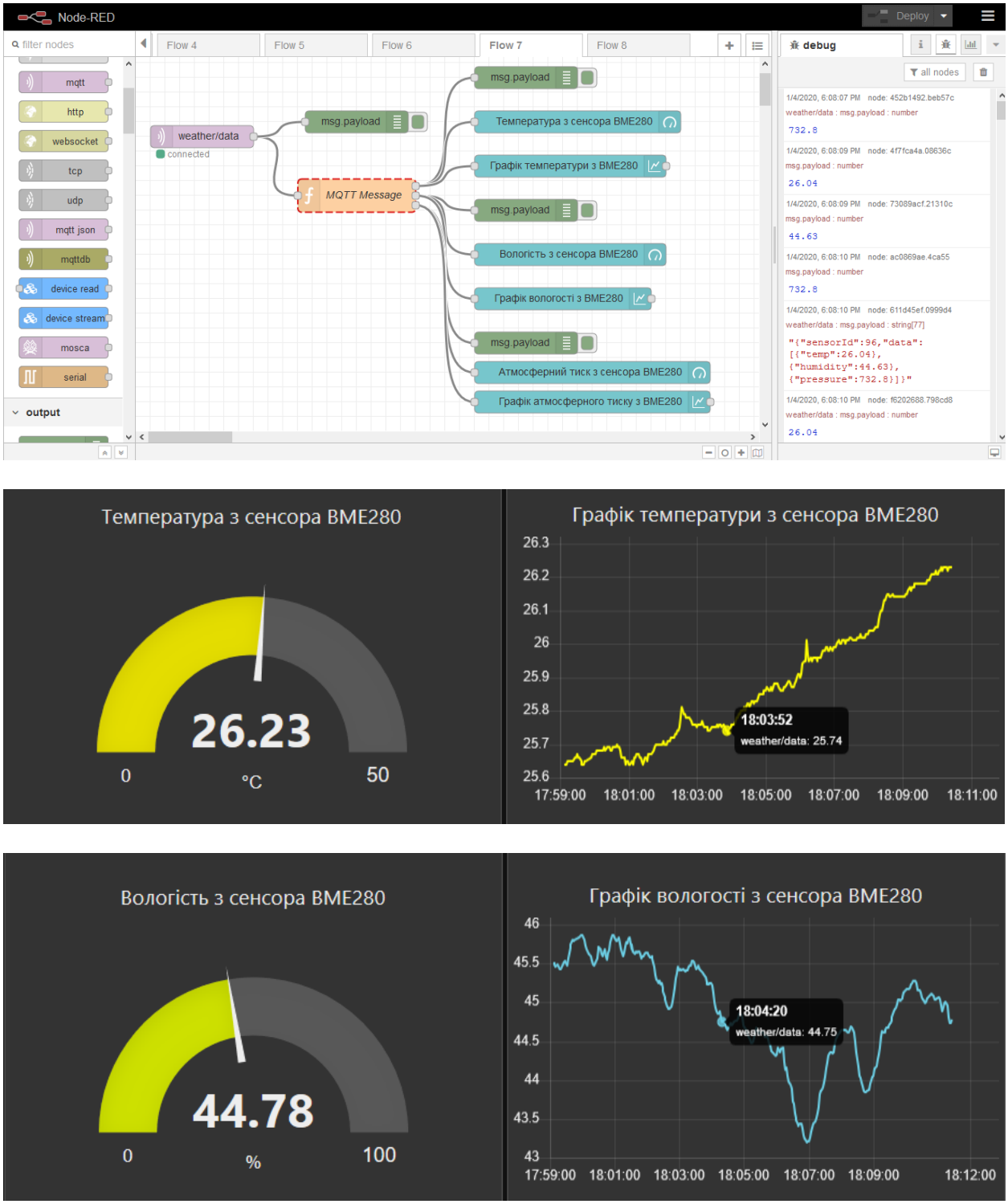



Рис.5.5. Вивід інформації на Node-RED Dashboard

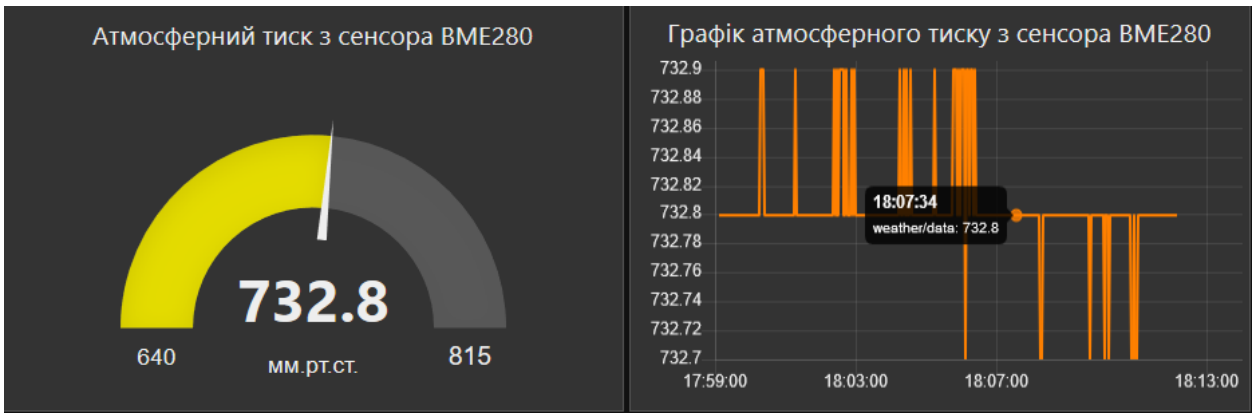
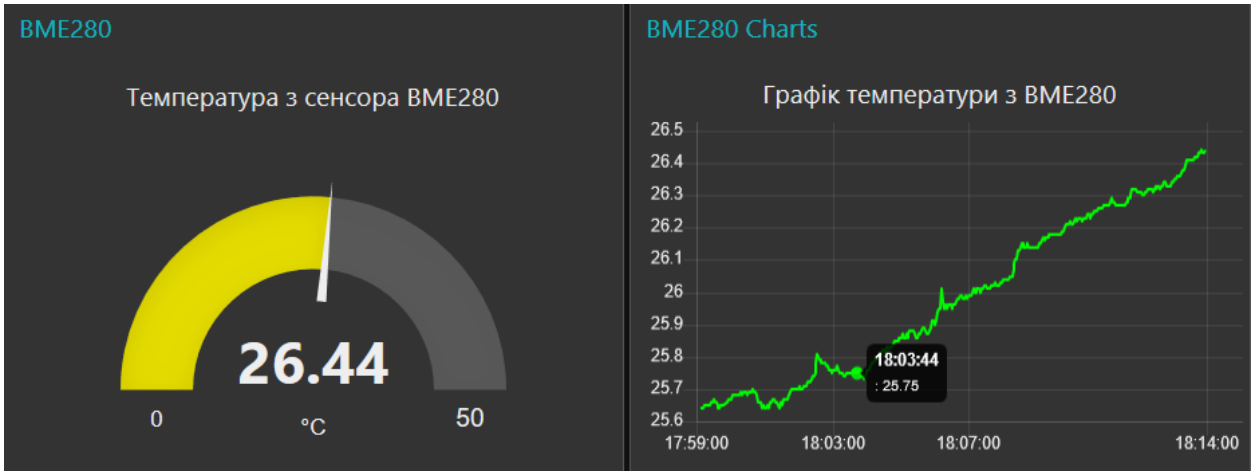
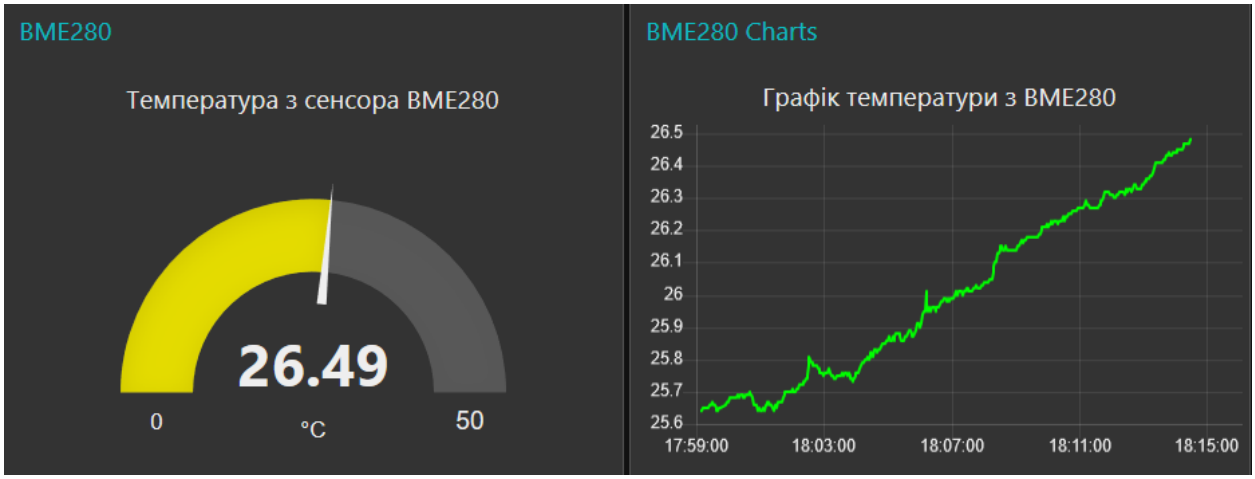



Рис.5.5. Вивід інформації на Node-RED Dashboard



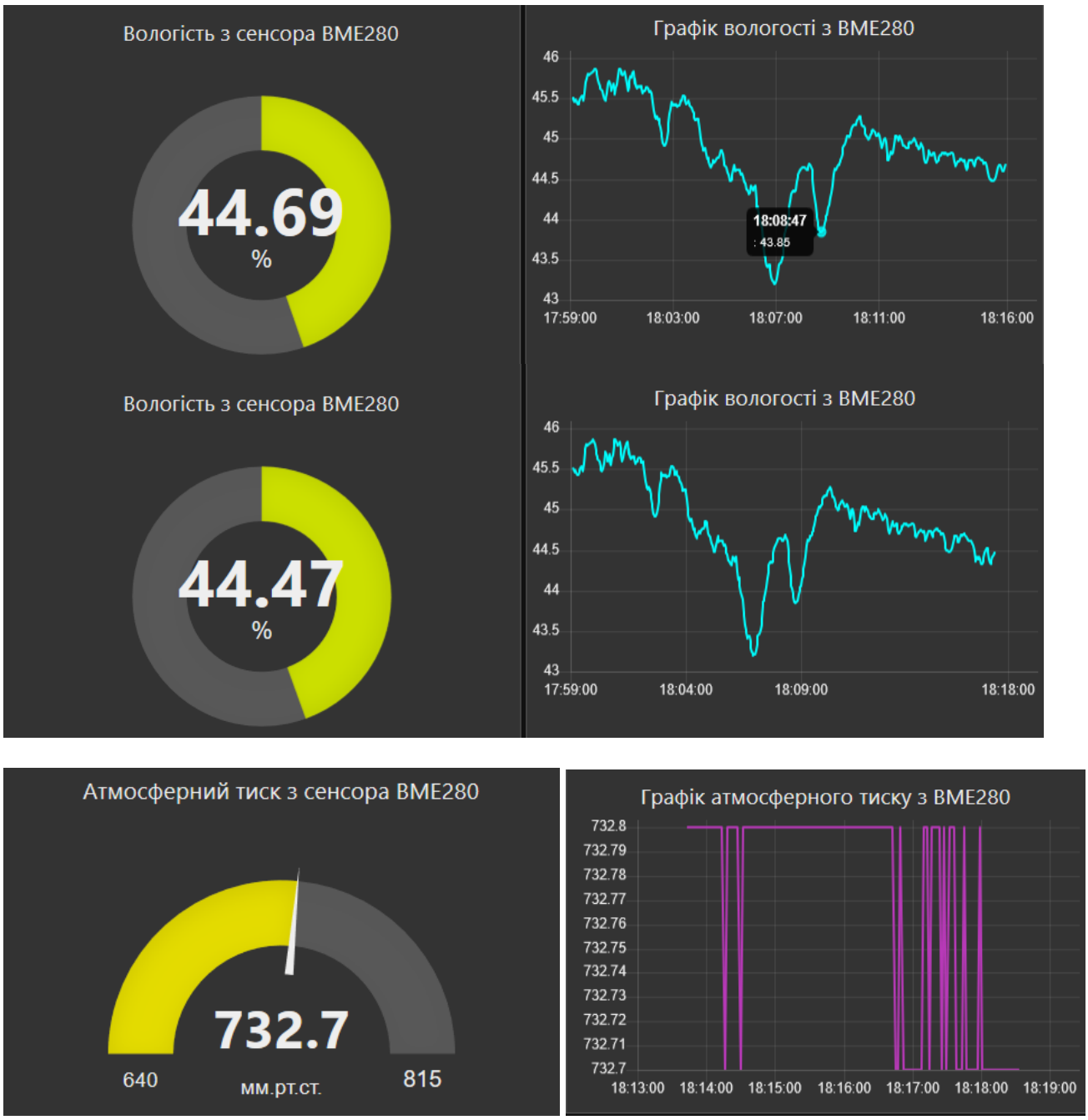
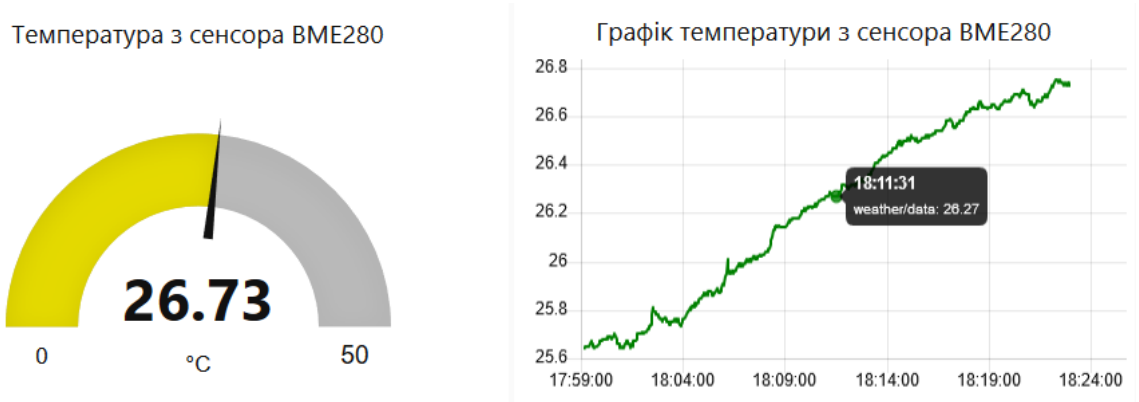
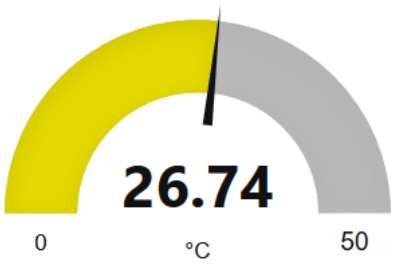


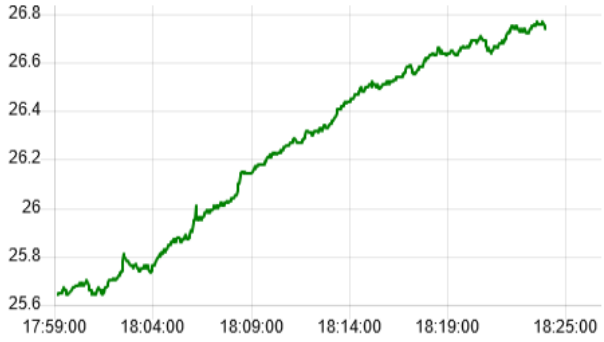

Рис.5.5. Вивід інформації на Node-RED Dashboard



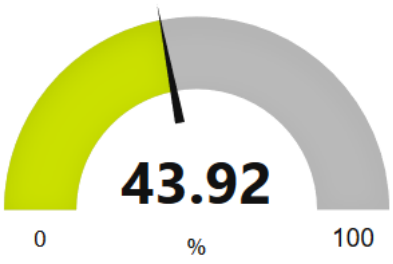
Температура з сенсора BME280



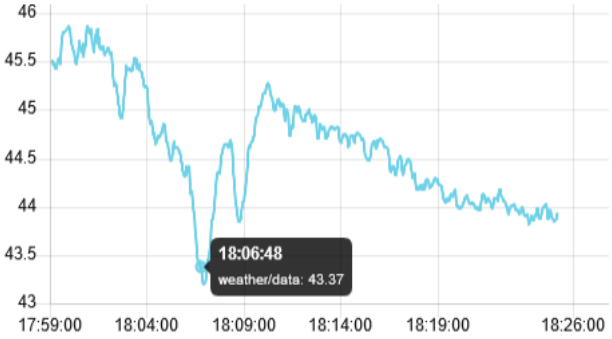

Графік температури з сенсора BME280



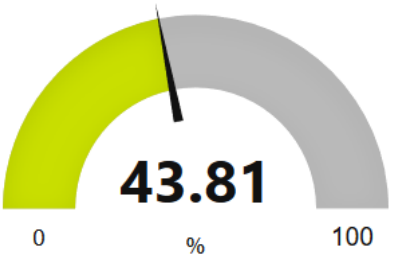

Вологість з сенсора BME280



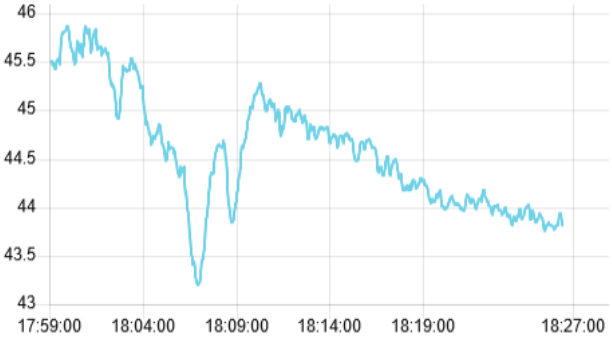

Графік вологості з сенсора BME280



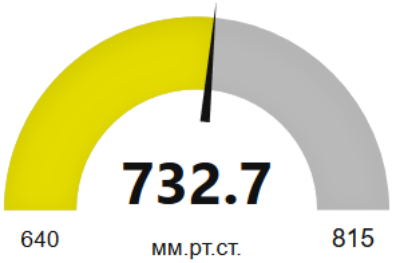

Вологість з сенсора BME280



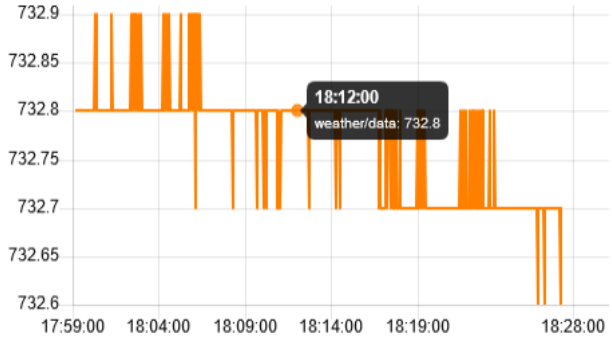

Графік вологості з сенсора BME280




Атмосферний тиск з сенсора BME280




Графік атмосферного тиску з сенсора BME280



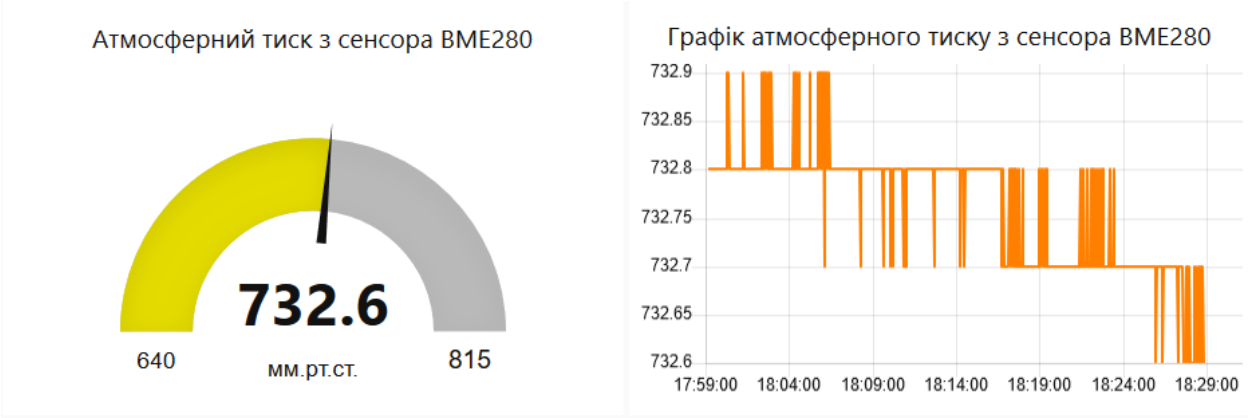
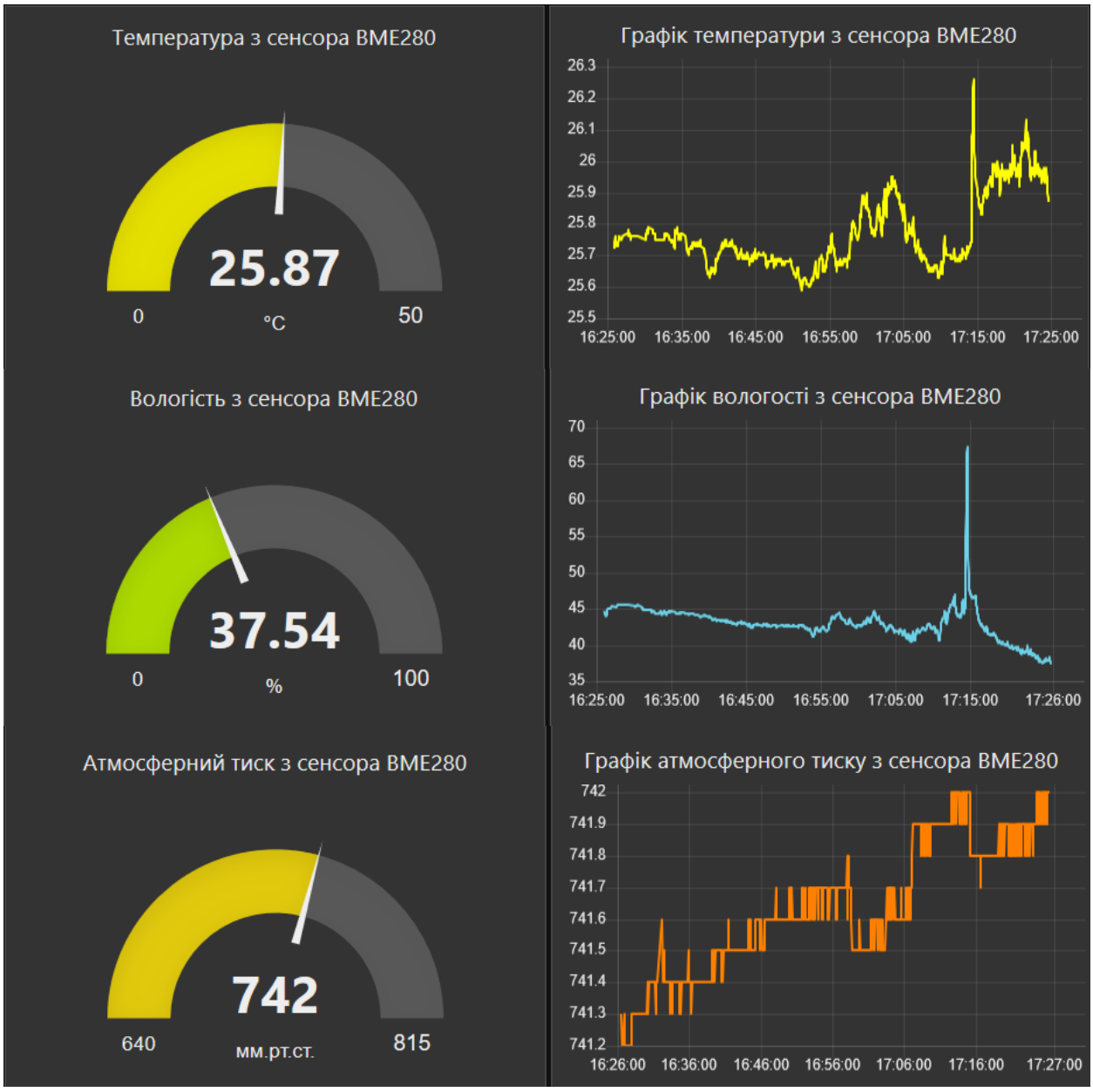



Рис.5.5. Вивід інформації на Node-RED Dashboard



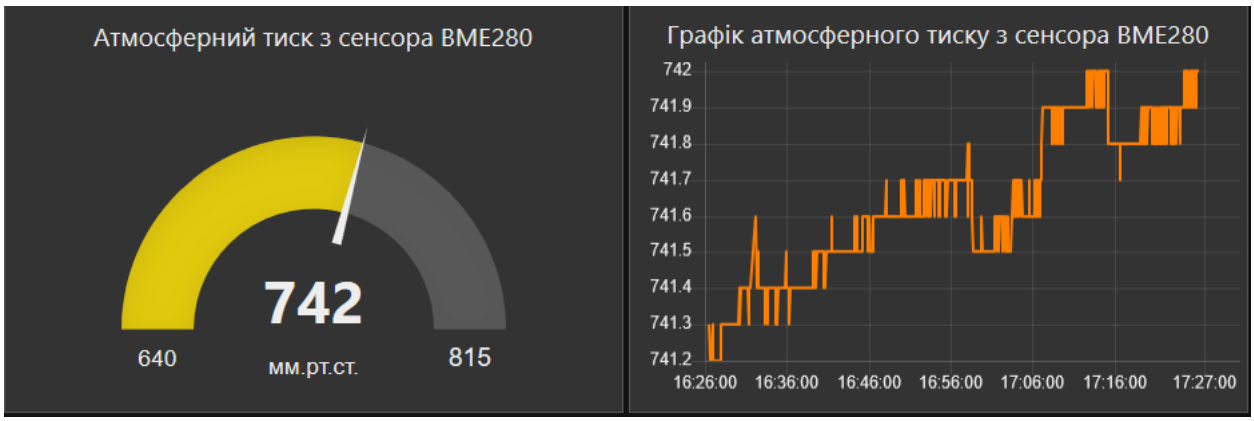
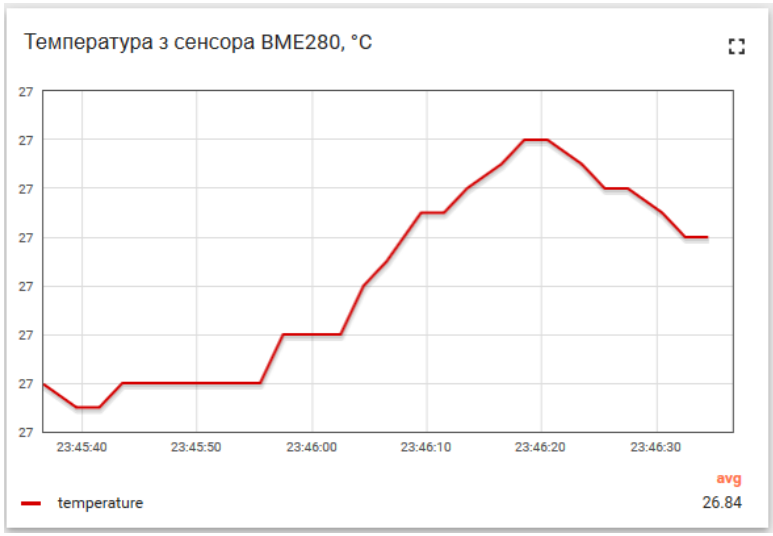
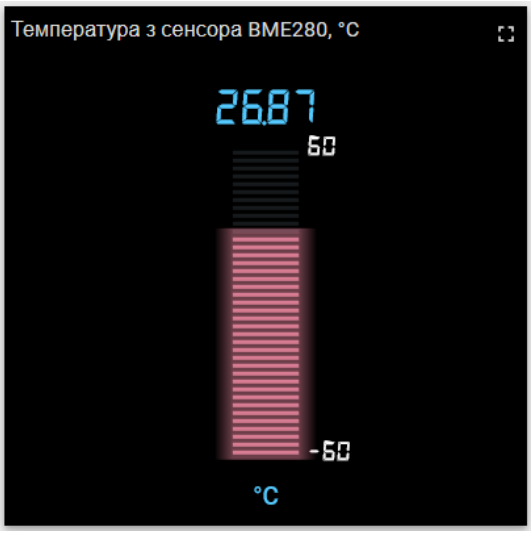
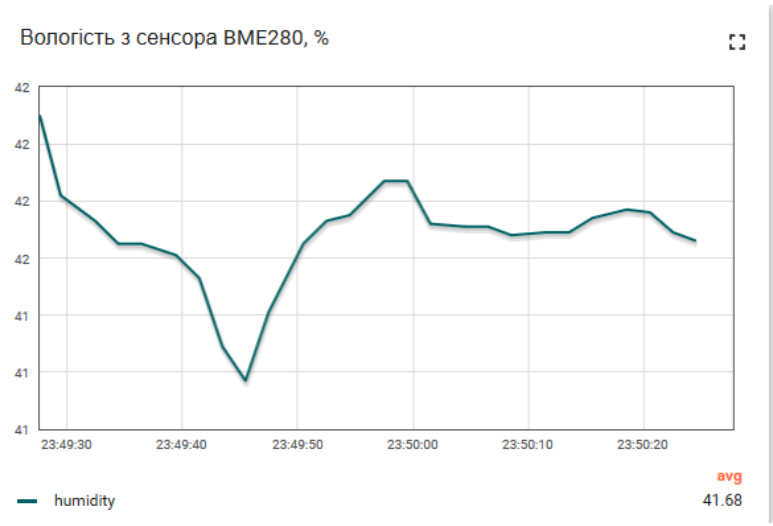
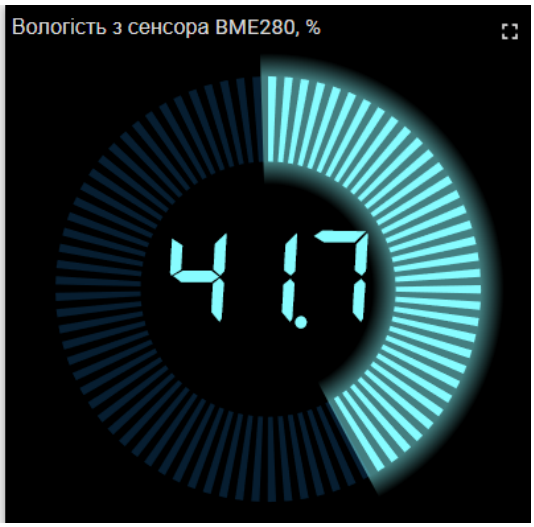


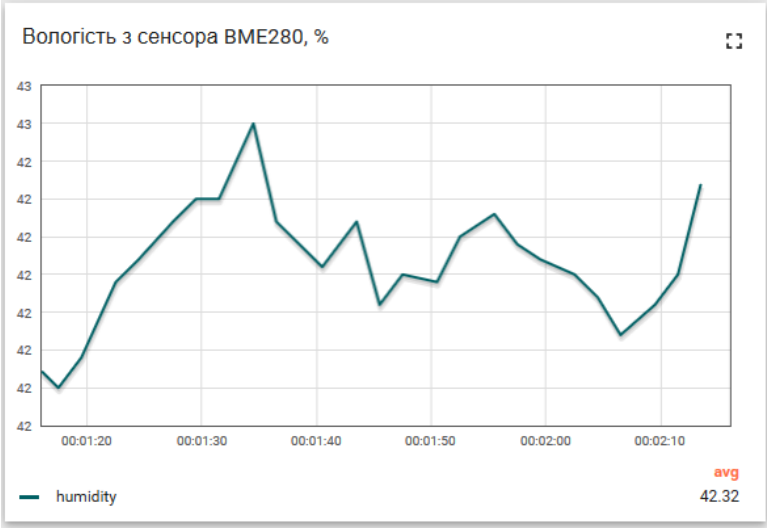

Рис.5.5. Вивід інформації на Node-RED Dashboard

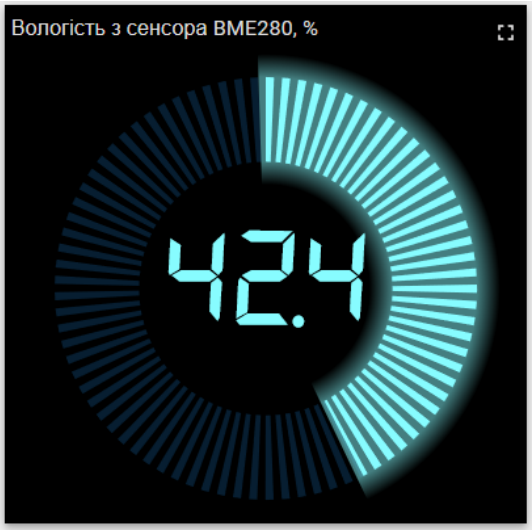


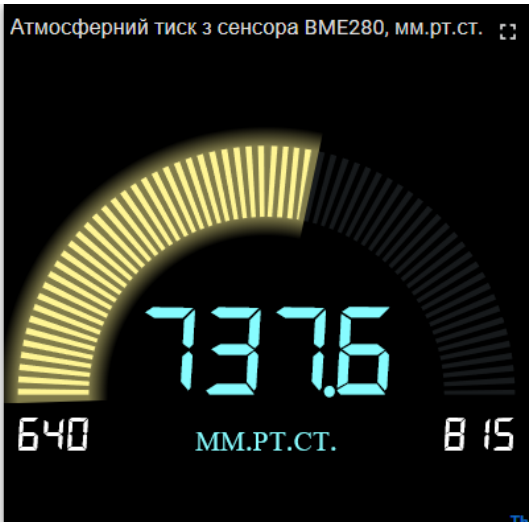



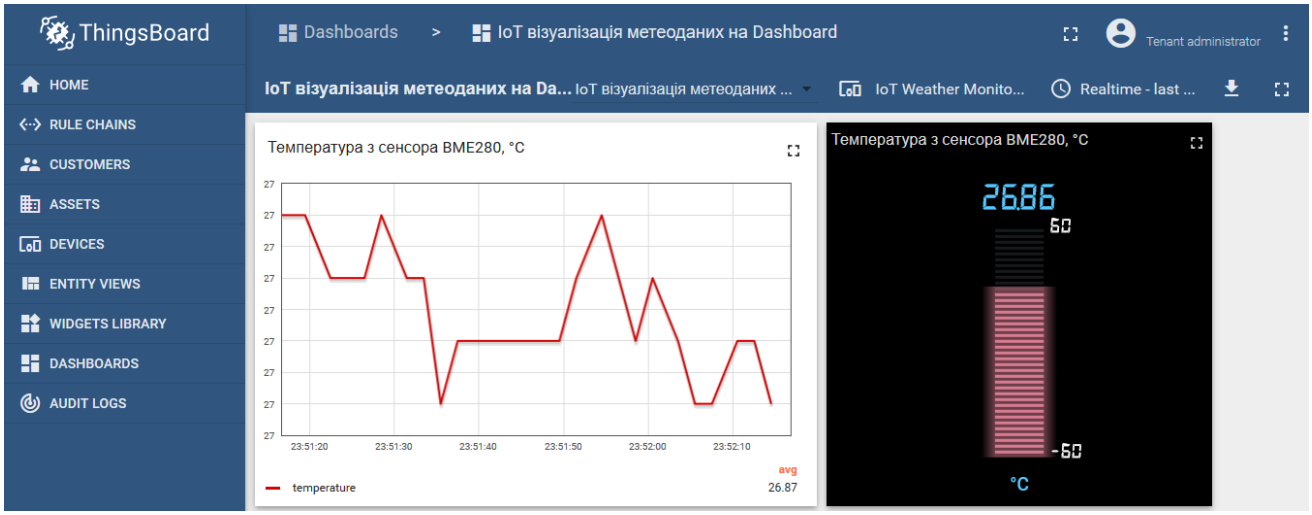



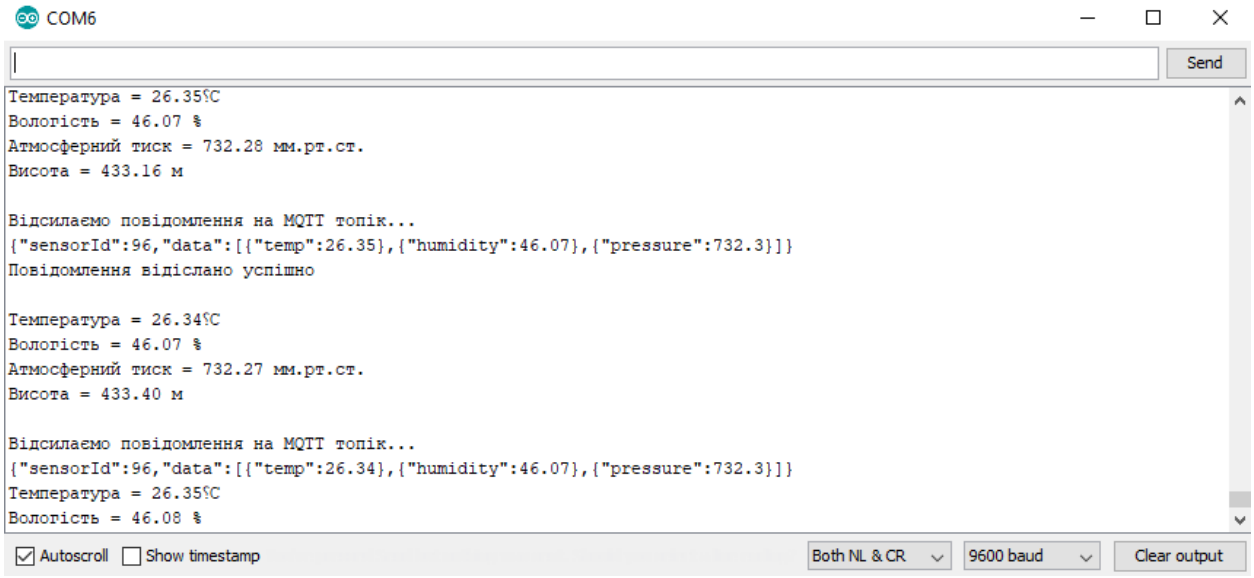

Вологість з сенсора BME280, %
43
43
42
42
42
42
42
42
42
42
00:01:20
00:01:30
00:01:40
00:01:50
00:02:00
00:02:10
avg
humidity
42.32


Вологість з сенсора BME280, %
42.4


Атмосферний тиск з сенсора BME280, мм.рт.ст.
738
738
738
738
738
738
738
738
23:50:40
23:50:50
23:51:00
23:51:10
23:51:20
23:51:30
avg
pressure
737.64


Атмосферний тиск з сенсора BME280, мм.рт.ст.
737.6
640
ММ.РТ.СТ.
8 15


ThingsBoard
Dashboards
>
IoT візуалізація метеоданих на Dashboard
Tenant administrator
HOME
RULE CHAINS
CUSTOMERS
ASSETS
DEVICES
ENTITY VIEWS
WIDGETS LIBRARY
DASHBOARDS
AUDIT LOGS
IoT візуалізація метеоданих на Da...
IoT візуалізація метеоданих ...
IoT Weather Monito...
Realtime - last ...
Температура з сенсора BME280, °C
27
27
27
27
27
27
27
27
23:51:20
23:51:30
23:51:40
23:51:50
23:52:00
23:52:10
avg
temperature
26.87
Температура з сенсора BME280, °C
26.86
60
-60
°C


COM6
Send
Температура = 26.35°C
Вологість = 46.07 %
Атмосферний тиск = 732.28 мм.рт.ст.
Висота = 433.16 м
Відсилаємо повідомлення на MQTT топік...
{"sensorId":96,"data":[{"temp":26.35},{"humidity":46.07},{"pressure":732.3}]}
Повідомлення відіслано успішно
Температура = 26.34°C
Вологість = 46.07 %
Атмосферний тиск = 732.27 мм.рт.ст.
Висота = 433.40 м
Відсилаємо повідомлення на MQTT топік...
{"sensorId":96,"data":[{"temp":26.34},{"humidity":46.07},{"pressure":732.3}]}
Температура = 26.35°C
Вологість = 46.08 %
Autoscroll
Show timestamp
Both NL & CR
9600 baud
Clear output

# ВИСНОВКИ

В роботі розроблено апаратно – програмні засоби моніторингу метеопараметрів з використанням технологій Інтернету Речей (IoT).

Апаратні засоби моніторингу метеопараметрів складаються з мікроконтролерного пристрою, що вимірює параметри погоди, такі як температура, відносна вологість повітря, атмосферний тиск та передає метеодані на IoT - платформу для подальшої обробки та візуалізації.

IoT - пристрій розроблено на платі Arduino Mega2560 з МК AVR ATmega2560, цифровому сенсорі атмосферного тиску, температури і вологості BME280, Wi-Fi модулі ESP-01 на чіпі ESP8266 та модулі РК - дисплею 16×2 HD44780. Мікроконтролерний пристрій моніторингу метеопараметрів має меню, що дозволяє користувачу вибрати Wi-Fi точку доступу, MQTT-сервер (IoT-платформу) та формат представлення даних.

Розроблено електричну принципову схему та модель IoT – пристрою моніторингу метеопараметрів в САПР Proteus VSM. Розроблено алгоритм роботи та програмне забезпечення IoT - пристрою моніторингу метеопараметрів. Програмне забезпечення складається з програмних модулів для роботи з цифровим сенсором BME280, модулем Wi-Fi ESP-01 та основної програми для МК AVR ATmega2560 плати Arduino Mega2560, що збирає, обробляє метеодані та передає їх по протоколу MQTT на локальний сервер Mosquitto або віддалену IoT – платформу ThingsBoard. Розроблено панелі візуалізації отриманих IoT-даних з локального сервера Mosquitto засобами Node-RED, а на віддаленій IoT – платформі ThingsBoard засобами ThingsBoard Dashboard.

Змодельовано роботу IoT - пристрою в емуляторі Proteus ISIS. Зібрано макет IoT - пристрою моніторингу метеопараметрів на макетній платі breadboard та протестовано його роботу.

Результати моделювання та тестування IoT - пристрою моніторингу метеопараметрів показали коректну роботу розробленого апаратного та програмного забезпечення.

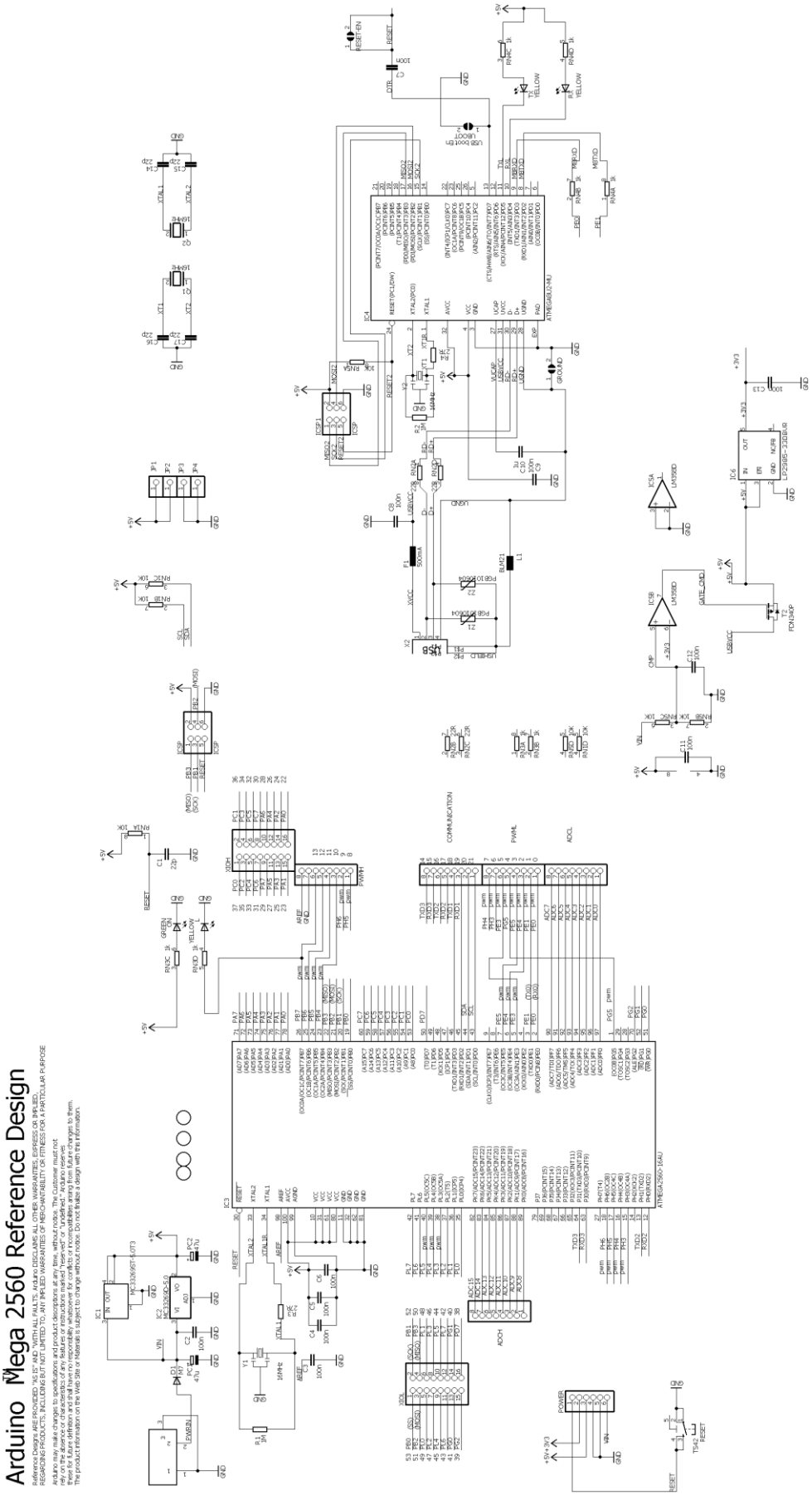
## СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

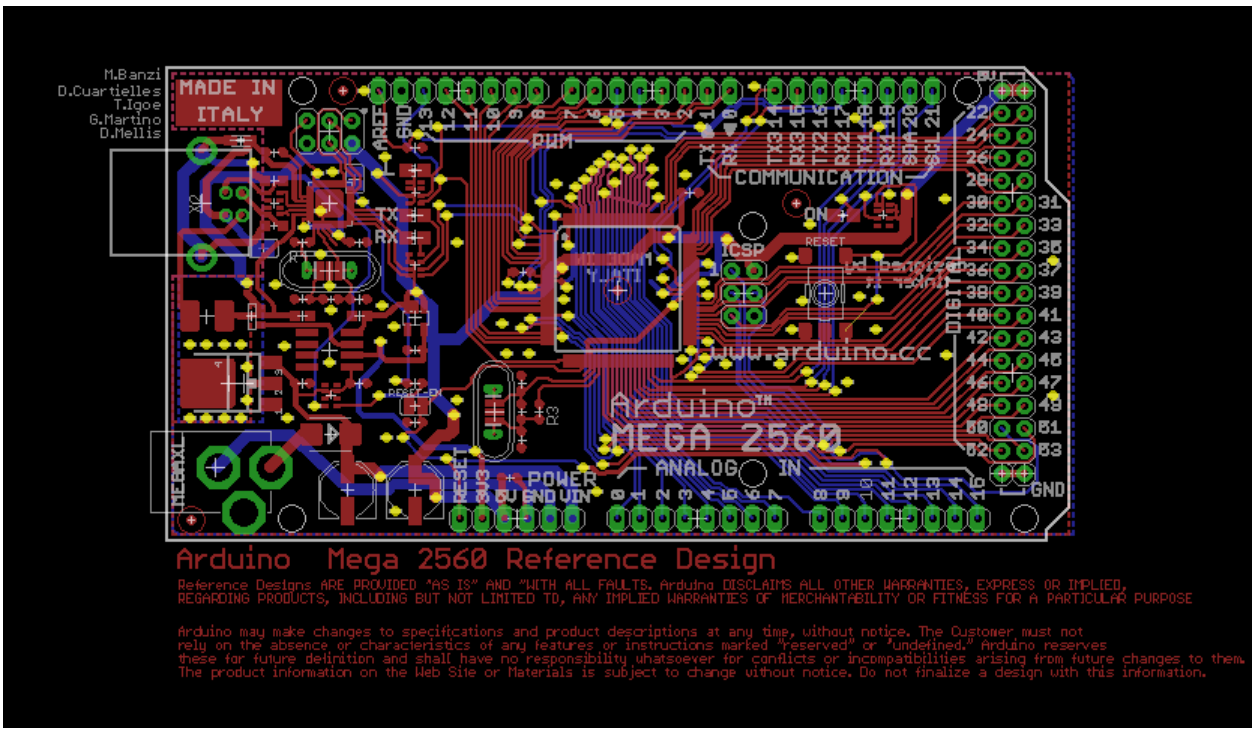

1. Петин В.А. 77 Проектов для Arduino. – ДМК-Пресс, 2019. – 356 с.: ил.
2. Бокселл Дж. Изучаем Arduino. 65 проектов своими руками. — СПб.: Питер, 2017. — 400 с.: ил..
3. Блум Джереми. Изучаем Arduino: инструменты и методы технического волшебства: Пер. с англ. — СПб.: БХВ-Петербург, 2015. — 336 с: ил.
4. С. Монк. Программируем Arduino. Профессиональная работа со скетчами СПб.: Питер, 2017. – 252 с.
5. Петин В.В. Arduino и Raspberry Pi в проектах Internet of Things. – БХВ – Петербург, 2016 – 320 с.
6. Макаров С.Л. Arduino Uno и Raspberry Pi 3. От схемотехники к интернету верей. – ДМК-Пресс, 2018 – 204 с.
7. Dexter Pearson. Arduino: 2019 Beginner’s Guide to Learn Arduino Programming Step by Step. – July 29, 2019, p. 61, ISBN-10: 1086093771.
8. Ethan Thorpe. Arduino for Beginners: Comprehensive Beginners Guide to Learn Arduino Programming Step by Step. – July 19, 2019 – p. 150.
9. Massimo Banzi, Michael Shiloh. Getting Started with Arduino: The Open Source Electronics Prototyping Platform (Make) 3rd Edition - Make Community, December 28, 2014 – p. 262.
10. Peter Waher, Pradeeka Seneviratne, Brian Russel, Drew Van Duren. IoT: Building Arduino-Based Projects – Packt Publishing, August 31, 2016 – p. 732.
11. Adeel Javed. Building Arduino Projects For The Internet Of Things: Experiments With Real-World Applications – 2016 – p. 285.
12. Tim Pulver. Hands-On Internet of Things with MQTT: Build connected IoT devices with Arduino and MQ Telemetry Transport (MQTT) – Packt Publishing; 1 edition (October 4, 2019) – p. 350.
13. Rajesh Singh, Anita Gehlor, Lovi Raj Gupta, Bhupendra Singh, and Mahendra Swain. Internet of Things with Raspberry Pi and Arduino. – CRC Press Taylor & Francis Group, LLC, 2020 – p. 190.

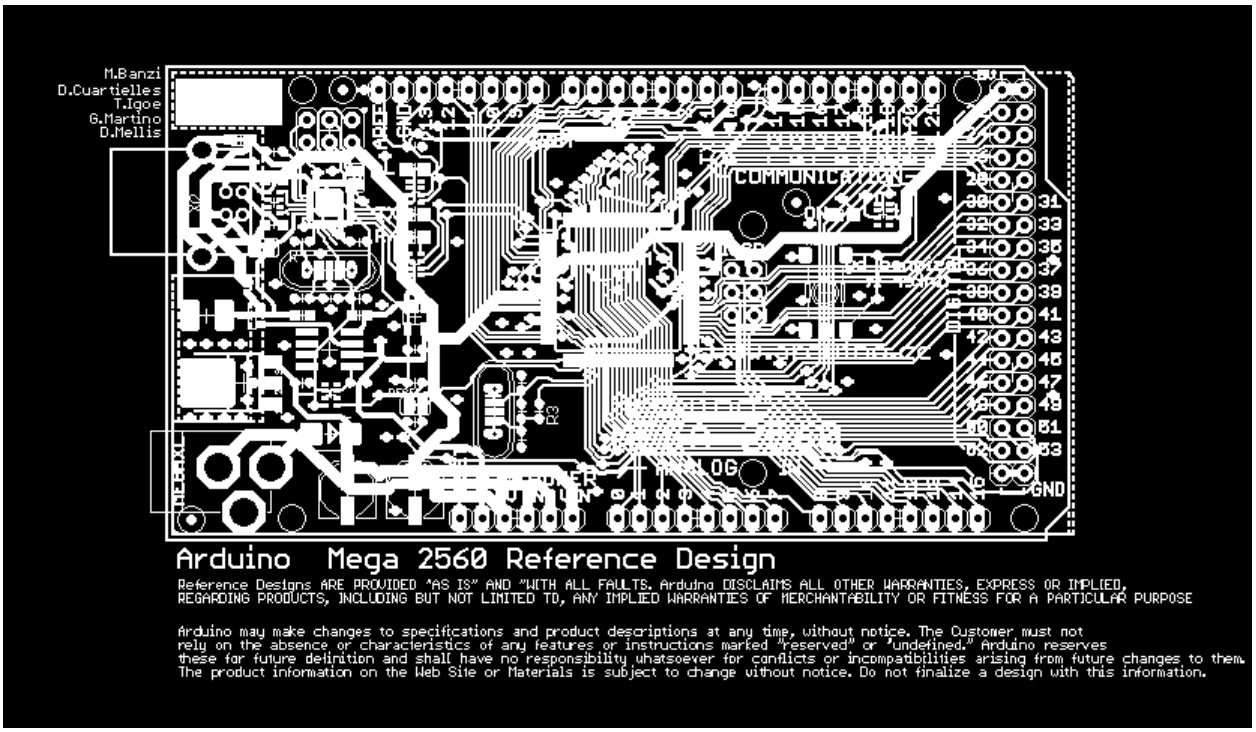
14. Emily Gertz & Patrick Di Justo. Environmental Monitoring with Arduino. Building Simple Devices to Collect Data About the World Around Us. – Published by O'Really Media Inc., 2012 – p. 96.
15. Середовище розробки Arduino IDE. Електронний ресурс: https://www.arduino.cc/en/Main/Software.
16. Електронний ресурс по точково-матричному LCD контролеру/драйверу Hitachi HD44780: https://circuitdigest.com/sites/default/files/HD44780U.pdf.
17. Електронний ресурс веб-аплікації генерування користувацьких шаблонів символів LCD: https://maxpromer.github.io/LCD-Character-Creator/.
18. Електронний ресурс datasheet по сенсору атмосферного тиску, температури і вологості BME280: http://www.embeddedadventures.com/datasheets/-BME280.pdf.
19. Електронний ресурс по сенсору BME280: https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/.
20. Електронний ресурс з описом підключення Wi-Fi модуля ESP8266 до плат Arduino та їх програмування: http://wiki.amperka.ru/%D0%BF%D1%80%-D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%8B:esp8266-wifi-module.
21. Електронний ресурс з описом протоколу MQTT: https://ipc2u.ru/articles/-prostye-resheniya/chto-takoe-mqtt/.
22. Електронний ресурс з описом протоколу MQTT: http://mqtt.org/documentation.
23. Електронний ресурс з описом протоколу MQTT: http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/.
24. Електронний ресурс з описом обміну даними по протоколу MQTT з використанням Arduino, ESP8266 і бібліотеки PubSubClient: https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/.
25. Елекронний ресурс з описом підключення Arduino до MQTT брокера: https://techtutorialsx.com/2017/04/09/esp8266-connecting-to-mqtt-broker/, https://iotbyhvm.ooo/arduino-pubsubclient-arduino-client-for-mqtt/, http://www.arjunsk.com/iot/esp8266-programming-cloud-mqtt-part-3/.

26. Електронний ресурс по IoT платформі ThingsBoard: https://thingsboard.io/-docs/user-guide/install/installation-options/?ceInstallType=liveDemo.
27. Електронний ресурс “Заватаження даних температури по протоколу MQTT на сервер ThingsBoard використовуючи Arduino Uno, ESP8266 і сенсор DHT22: https://thingsboard.io/docs/samples/arduino/temperature/
28. Електронний ресурс по MQTT брокеру Mosquitto: https://mosquitto.org/.
29. Електронний ресурс по Node-RED: https://nodered.org/docs/getting-started/windows.

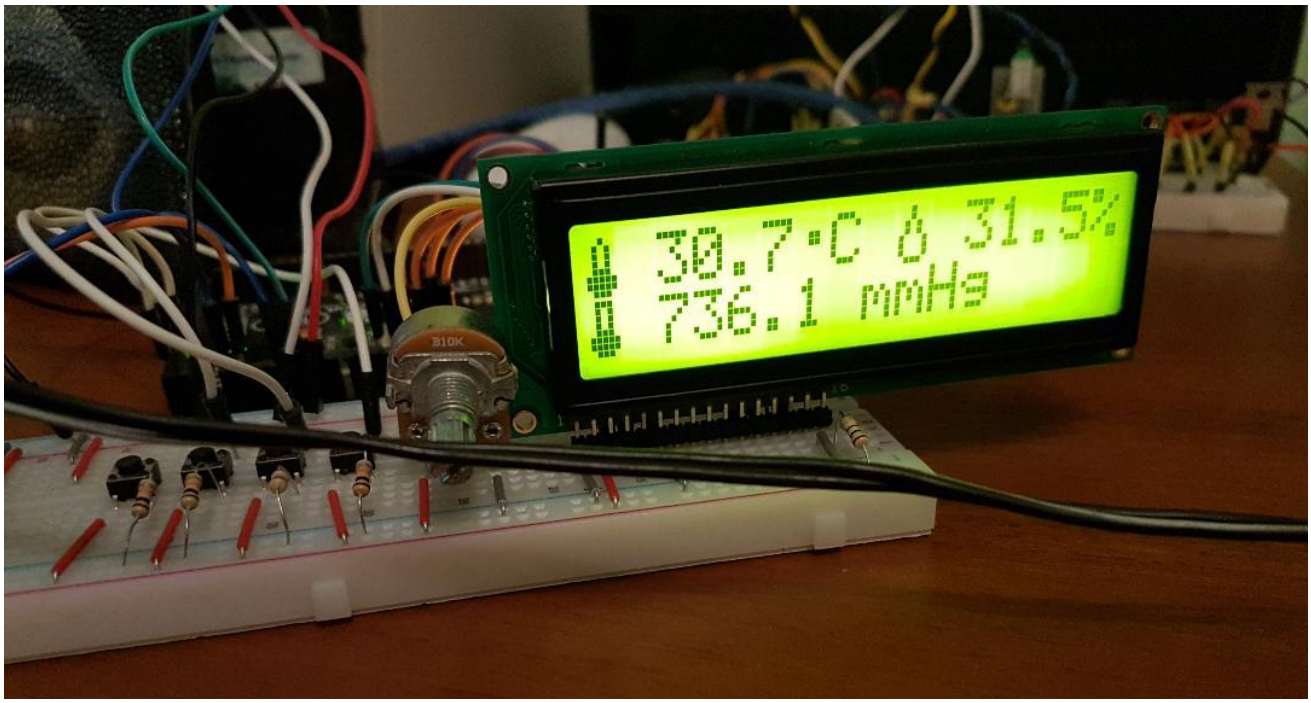
# ДОДАТКИ

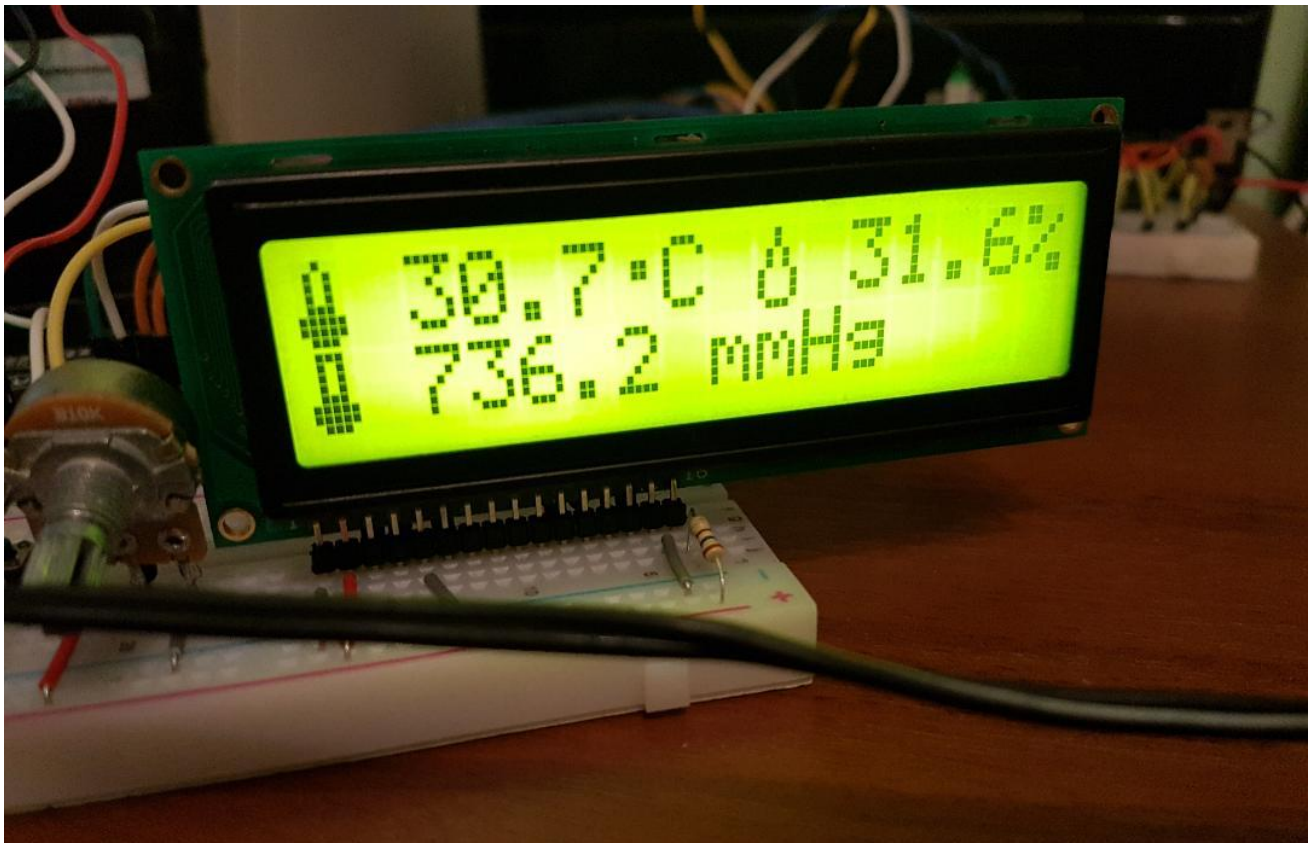

Arduino Mega 2560 Reference Design
Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for its future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

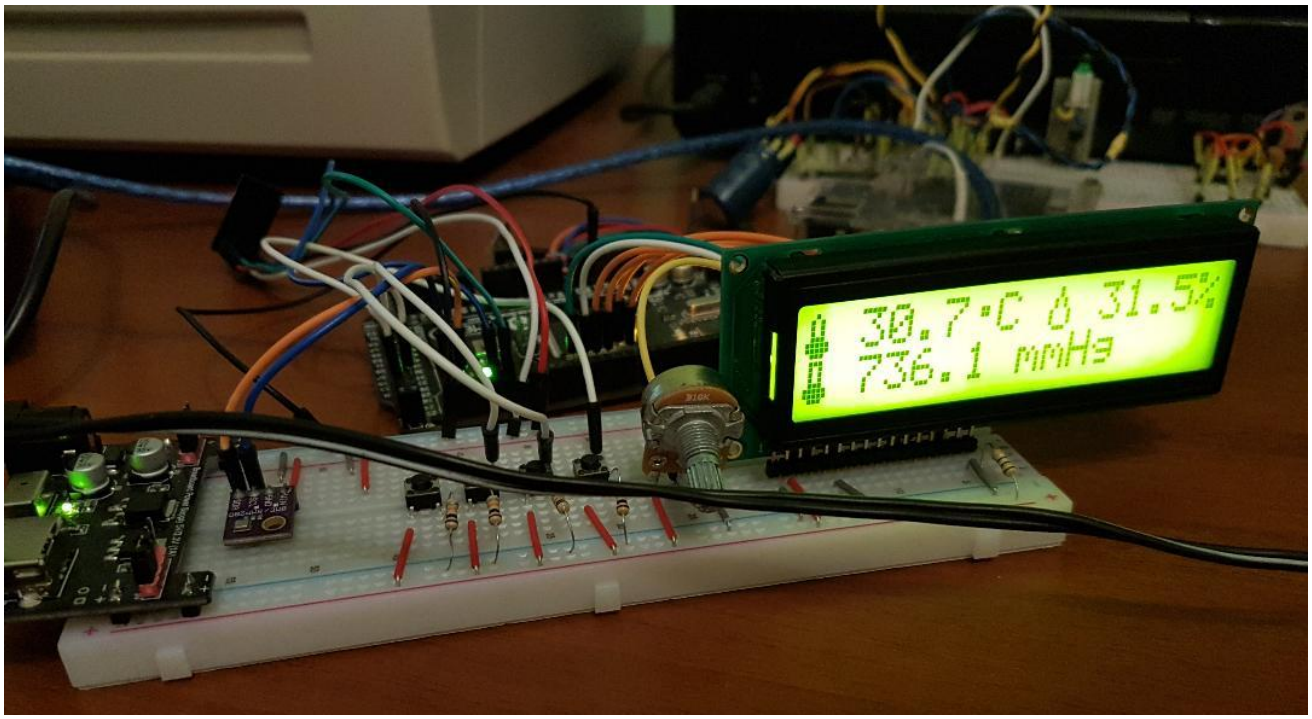

M.Banzi
D.Cuartielles
T.Igoe
G.Martino
D.Mellis
MADE IN
ITALY
PWM
COMMUNICATION
ICSP
RESET
www.arduino.cc
Arduino™
MEGA 2560
POWER
ANALOG IN
Arduino Mega 2560 Reference Design
Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED,
REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not
rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves
these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

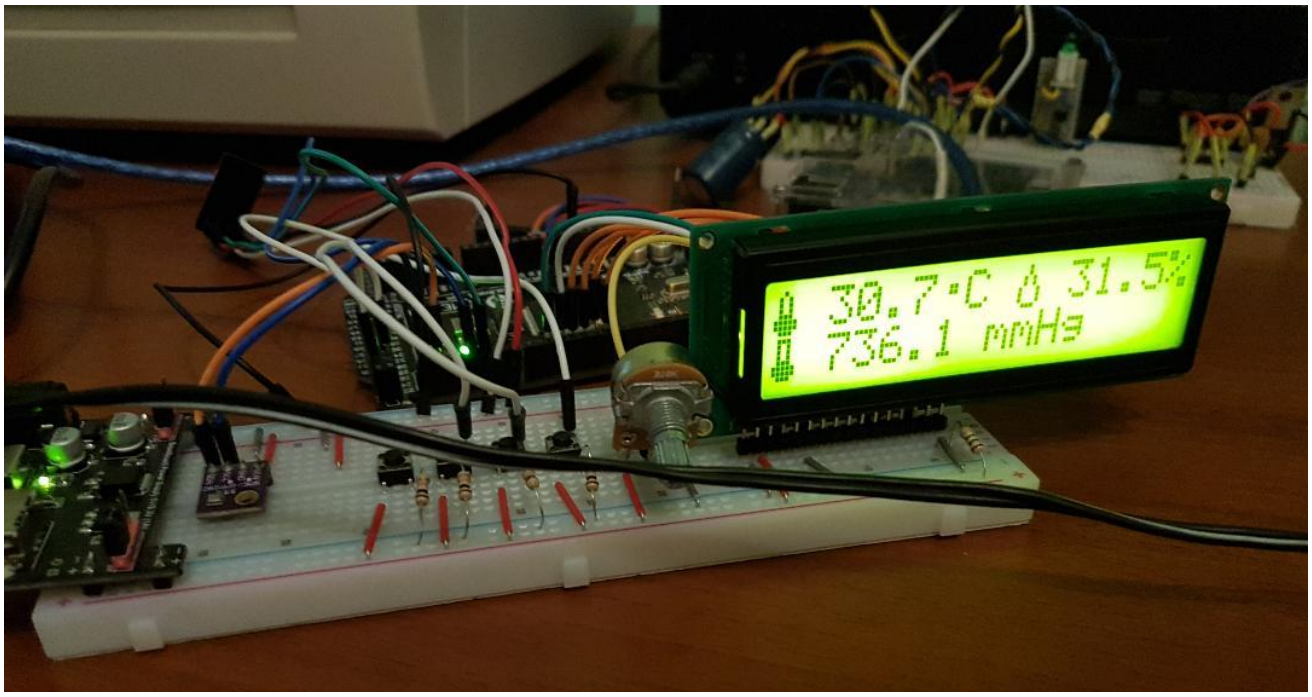

M.Banzi
D.Cuartielles
T.Igoe
G.Martino
D.Mellis
Arduino Mega 2560 Reference Design
Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED,
REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not
rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves
these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

**Додаток. Макет IoT-пристрою моніторингу метеопараметрів на платі Arduino Mega2560, цифровому сенсорі атмосферного тиску, температури і вологості BME280, Wi-Fi модулі ESP8266.**




30.7°C 31.5%
736.1 mmHg


30.7°C 31.5%
736.1 mmHg

## Додаток. Головний програмний модуль IoT - пристрою моніторингу метеопараметрів

```
/*
  IoT - пристрій моніторингу метеопараметрів з використанням платформи Arduino та Wi-Fi модуля ESP8266
  (IoT based weather monitoring device using Arduino and ESP8266 Wi-Fi module)
  Розробив: студент групи КН-61М, Олег Кубрак
*/

#include <Wire.h> // бібліотека для роботи з різними пристроями по інтерфейсу I2C/TWI
#include <Adafruit_Sensor.h> // бібліотека-драйвер для роботм з різними сенсорами
#include <Adafruit_BME280.h> // бібліотека для роботи з сенсором температури, вологості і тиску BME280
#include <LiquidCrystalRus.h> //<LiquidCrystal.h>
#include <ArduinoJson.h> // бібліотека для роботи з форматом JSON
#include <WiFiEsp.h> // бібліотека для роботи з Wi-Fi модулем ESP8266
#include <WiFiEspClient.h>
#include <WiFiEspUdp.h>
#include <PubSubClient.h> // MQTT-клієнт "Publisher-Subscriber"

// ініціалізуємо кнопки
#define NUM_OF_BTNS 4
#define MENU_BTN_PIN 19
#define SELECT_PLUS_BTN_PIN 23
#define SELECT_MINUS_BTN_PIN 22
#define EXIT_BTN_PIN 24

unsigned long lastDebounceTime[NUM_OF_BTNS] = {0, 0, 0, 0}; // останній час, коли вихідний пін був переключений

int lastBtnState[NUM_OF_BTNS] = {LOW, LOW, LOW, LOW};
int btnState[NUM_OF_BTNS];
int btnsArray[NUM_OF_BTNS] = {MENU_BTN_PIN, SELECT_PLUS_BTN_PIN, SELECT_MINUS_BTN_PIN, EXIT_BTN_PIN};

const char *WiFi_ssid[] = {"GoldNew", "ASUS_Network", "TP_LINK_49F6F2"}; // назва Wi-Fi точки доступу
const char *WiFi_passwd[] = {"qwerty12345678", "12345678Lv", "Test12345678"}; // пароль до Wi-Fi точки доступу

int status = WL_IDLE_STATUS;
unsigned long lastSend;

const char *mqtt_server[] = {"192.168.0.104" /* Mosquitto broker */, "demo.thingsboard.io" /* сервіс ThingsBoard */};
const int mqtt_port = 1883;
const char *token[] = {"", "BME280_SENSOR_TOKEN"}; // BME280_SENSOR_TOKEN

const char *topic[] = {"weather/data", "v1/devices/me/telemetry"};
```

```
float temp, humidity, pressure, altetude;
char jsonBuffer[512];
size_t jsonMessageSize;

// створюємо та ініціалізуємо об'єкти WiFiEspClient, PubSubClient
WiFiEspClient espClient;
PubSubClient client(espClient);

// створюємо значки температури, вологості і тиску для LCD
byte tempChar[] = { // lcd значок/іконка температури
  0x04,
  0x0A,
  0x0A,
  0x0A,
  0x0E,
  0x1F,
  0x1F,
  0x0E
};
byte humidityChar[] = { // lcd значок вологості
  0x04,
  0x04,
  0x0A,
  0x0A,
  0x11,
  0x11,
  0x11,
  0x0E
};
byte pressureChar[] = { // lcd значок тиску
  0x0E,
  0x0A,
  0x0A,
  0x0A,
  0x0E,
  0x1F,
  0x1F,
  0x0E
};
//ініціалізуємо бібліотеку номерами пінів плати Arduino до яких підключені відповідні інтерфейсні піни LCD
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystalRus lcd(rs, en, d4, d5, d6, d7);

const byte interruptPin = 19; // interrupt0 - pin 2, interrupt1 - pin 3, interrupt2 - pin 21, interrupt3 - pin 20, interrupt4 - pin 19, interrupt5 - pin 18
volatile bool mainMenuInvoke = false;
unsigned long debounceDelay = 50, delayTime;    // час деренчання; збільшити якщо вихід деренчить (шумить), час затримки
int wifi_ind = 0, mqtt_server_ind = 0;

#define SEA_LEVEL_PRESSURE_HPA (1028.00) // тиск над рівнем моря для Львова 1028 гПа
```

```
Adafruit_BME280 bme; // підключення по I2C

/*const char iot_device_banner[] = "IoT based weather monitoring device";
const char author_info[] = "(C) 2021, developed by Oleh Kubryk";*/
const char iot_device_banner[] = "IoT пристрій моніторингу метеопараметрів";
const char author_info[] = "Розробив ст. КН-61М, Олег Кубрик";

void wifiInit()
{
  // ініціалізація послідовного інтерфейсу для модуля ESP8266
  Serial2.begin(9600);
  lcd.clear();
  //lcd.print("Initializing");
  lcd.print("Ініціалізація");
  lcd.setCursor(0,1);
  //lcd.print("ESP module..");
  lcd.print("ESP модуля...");
  // ініціалізація модуля ESP8266
  WiFi.init(&Serial2);
  lcd.print("OK!");
  delay(100);
  lcd.clear();
  lcd.home();
  // перевірка чи підключений Wi-Fi шилд
  if (WiFi.status() == WL_NO_SHIELD) {
    //Serial.println(F("WiFi шилд відсутній"));
    Serial.println("Wi-Fi shield not present");
    lcd.print("Wi-Fi shield");
    //lcd.print("Wi-Fi шилд");
    lcd.setCursor(0,1);
    //lcd.print("not present");
    lcd.print("не знайдено");
    while (true); // не продовжуємо
  }
  Serial.println("Підключаємося до Wi-Fi точки доступу ...");
  //Serial.println("Connecting to AP ...");
  //lcd.print("Connecting to");
  lcd.print("Підключення до ");
  lcd.setCursor(0,1);
  lcd.print(WiFi_ssid[wifi_ind]);
  delay(100);
  // спроба підключення до мережі WiFi
  while (status != WL_CONNECTED) {
    Serial.print(F("Спроба підключитися до WPA SSID: "));
    //Serial.print("Attempting to connect to WPA SSID: ");
    //lcd.clear();
    //lcd.print("WPA SSID...");
    Serial.println(WiFi_ssid[wifi_ind]);
    // підключення до мережі WPA/WPA2
    status = WiFi.begin(WiFi_ssid[wifi_ind], WiFi_passwd[wifi_ind]);
    delay(500);
```

```
  }
  if (status == WL_CONNECTED) {
    Serial.println(F("Підключено до AP"));
    //Serial.println(F("Connected to AP"));
    lcd.clear();
    //lcd.setCursor(0,1);
    //lcd.print("connected to AP");
    lcd.print("Підключено");
    lcd.setCursor(0,1);
    lcd.print("до AP!");
    delay(100);
  }
}

void reconnect()
{
  lcd.clear();
  // цикл до тих пір поки не перепідключимося
  while (!client.connected()) {
    Serial.print("Підключаємося до MQTT-сервера...");
    lcd.setCursor(0,0);
    //lcd.print("Connect to MQTT");
    lcd.print("Підключення");
    lcd.setCursor(0,1);
    lcd.print("до сервера...");
    // Attempt to connect (clientId, username, password)
    if ( client.connect("mqttClient", token[mqtt_server_ind], NULL) ) {
      Serial.println( F("done") );
      lcd.print("OK!");
      delay(100);
    } else {
      Serial.print( F("failed, rc = " ));
      int state = client.state();
      Serial.print(state);
      lcd.setCursor(0,1);
      lcd.print("failed, rc="); lcd.print(state); lcd.print("   ");
      Serial.println(F(": нова спроба підключення через 5 секунд]"));
      // очікування 5 секунд перед новою спробою підключення
      delay(5000);
      //lcd.clear();
    }
  }
  //lcd.clear();
}

void infoAboutDeviceOutput()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(iot_device_banner);
  lcd.setCursor(0, 1);
  lcd.print(author_info);
```

```
  for(int posCount = 0; posCount < 23; posCount++)
  {
    lcd.scrollDisplayLeft(); //builtin command to scroll left the text
    delay(300); // затримка на 300 мс
  }
  lcd.clear();
}

// button array pos
int btnToPos(uint8_t btn)
{
   for (int i = 0; i < NUM_OF_BTNS; i++) {
     if (btnsArray[i] == btn)
       return i;
   }
   return -1;
}

bool isBtnPressed(int btn_pin)
{
  // читаємо пін кнопки, якщо кнопка натиснута буде високим, якщо ні буде низьким
  int value = digitalRead(btn_pin);
  int pos = btnToPos(btn_pin);
  /*  випалку якщо зчитане значення не рівне останьому стану кнопки, то встановити останній
час деренчання (брязкотіння) у поточний час в мілісекундках,
  що пройшоа з початку викоання програми millis() */
  if (value != lastBtnState[pos])  {
    lastDebounceTime[pos] = millis();
  }
  // перевірити різницю між поточним часом і останнім зареєстрованим часом натиснення
кнопки, якшо він більший встановленого часу затримки, то означає що натиснута кнопка
  if ((millis() - lastDebounceTime[pos]) > debounceDelay) {
    if (value != btnState[pos]) {
      btnState[pos] = value;
      if (btnState[pos] == HIGH)
        return true;
    }
  }
  //зберегти зчитане значення стан кнопки. наступного ращу в циклі зчитаний стан кнопки буде
вже як lastButtonState
  lastBtnState[pos] = value;
  return false;
}

void setWiFiAP()
{
  const char WiFiAP_MenuTitle[] = "* Wi-Fi AP *";
  int i = 0;
  lcd.clear();
  lcd.setCursor(2,0);
  lcd.print(WiFiAP_MenuTitle);
  lcd.setCursor(0,1);
```

```
  lcd.print("> ");
  lcd.setCursor(2,1);
  lcd.print(WiFi_ssid[i]);
  /*lcd.setCursor(15,1);
  lcd.print(">");*/
  while(1)
  {
    lcd.setCursor(2,1);
    lcd.print(WiFi_ssid[i]);

    if (isBtnPressed(SELECT_PLUS_BTN_PIN))
    {
      if (i == 2) i = 0;
      else i++;
      lcd.clear();
      lcd.setCursor(2,0);
      lcd.print(WiFiAP_MenuTitle);
      lcd.setCursor(0,1);
      lcd.print("> ");
      lcd.setCursor(2,1);
      lcd.print(WiFi_ssid[i]);
    }

    if (isBtnPressed(SELECT_MINUS_BTN_PIN))
    {
      if (i == 0) i = 2;
      else i--;
      lcd.clear();
      lcd.setCursor(2,0);
      lcd.print(WiFiAP_MenuTitle);
      lcd.setCursor(0,1);
      lcd.print("> ");
      lcd.setCursor(2,1);
      lcd.print(WiFi_ssid[i]);
    }

    if(isBtnPressed(MENU_BTN_PIN))
    {
      wifi_ind = i;
      return;
    }
    if (isBtnPressed(EXIT_BTN_PIN))
      return;
  }
}

void setMQTTServer()
{
  const char MQTTServerMenuTitle[] = "* MQTT-сервер *";
  const char *mqttServerName[] = {"Mosquitto" , "ThingsBoard"};
  //const uint8_t xpos[] = {2, 1};
  int i = 0;
```

```
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(MQTTServerMenuTitle);
  lcd.setCursor(0,1);
  lcd.print("> ");
  /*lcd.setCursor(15,1);
  lcd.print(">");*/
  lcd.print(mqttServerName[i]);

  while(1)
  {
    if (isBtnPressed(SELECT_PLUS_BTN_PIN))
    {
      if (i == 1) i = 0;
      else i++;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(MQTTServerMenuTitle);
      lcd.setCursor(0,1);
      lcd.print("> ");
      lcd.print(mqttServerName[i]);
    }
    if (isBtnPressed(SELECT_MINUS_BTN_PIN))
    {
      if (i == 0) i = 1;
      else i--;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(MQTTServerMenuTitle);
      lcd.setCursor(0,1);
      lcd.print("> ");
      lcd.print(mqttServerName[i]);
    }
    if(isBtnPressed(MENU_BTN_PIN))
    {
      mqtt_server_ind = i;
      client.setServer(mqtt_server[mqtt_server_ind], mqtt_port);
      return;
    }
    if (isBtnPressed(EXIT_BTN_PIN))
      return;
  }
}

void ISR_BtnPressed() {
  /*buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);.*/
  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (digitalRead(MENU_BTN_PIN) == HIGH) //isBtnPressed(MENU_BTN_PIN))
  {
     detachInterrupt(digitalPinToInterrupt(interruptPin));
     mainMenuInvoke = true;
```

```
  }
}

void mainMenu()
{
  /*char *mainMenuItems[] = {" Wi-Fi AP  ", "MQTT broker", "Data format", "   Exit    "};
  char menuTitle[] = "** Main Menu **";*/
  char *mainMenuItems[] = {"  Wi-Fi AP ", "MQTT-сервер ", "Формат даних", "   Вихід    "};
  char menuTitle[] = "* Головне Меню *";
  int i = 0;
  bool lcdPrint = false;
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(menuTitle);
  lcd.setCursor(2,1);
  lcd.print(mainMenuItems[i]);
  while(1)
  {
    if (lcdPrint)
    {
      lcd.setCursor(2,1);
      lcd.print(mainMenuItems[i]);
      lcdPrint = !lcdPrint;
    }
    if (isBtnPressed(SELECT_PLUS_BTN_PIN))
    {
      if(i == 3) i = 0;
      else i++;
      lcdPrint = !lcdPrint;
      //digitalWrite(27, HIGH); // turn the LED on (HIGH is the voltage level)
    }
    if (isBtnPressed(SELECT_MINUS_BTN_PIN))
    {
      if (i == 0) i = 3;
      else i--;
      lcdPrint = !lcdPrint;
      //digitalWrite(27, LOW); // turn the LED off by making the voltage LOW
    }
    if(isBtnPressed(MENU_BTN_PIN))
    {
      switch(i)
      {
        case 0: setWiFiAP(); break;
        case 1: setMQTTServer(); break;
        case 2: return;
        default: break;
      }
      lcd.clear();
      //lcd.setCursor(0,0);
      lcd.print(menuTitle);
      lcd.setCursor(2,1);
      lcd.print(mainMenuItems[i]);
```

```
    }
    if (isBtnPressed(EXIT_BTN_PIN))
    {
      lcd.clear();
      mainMenuInvoke = 0;
      attachInterrupt(digitalPinToInterrupt(interruptPin), ISR_BtnPressed, CHANGE);
      return;
    }
  }
}

void setup() {
    Serial.begin(9600);
    while(!Serial);  // time to get serial running
    lcd.begin(16, 2);  // виставляємо кількість стовпців і рядків LCD:
    lcd.createChar(0, tempChar);
    lcd.createChar(1, humidityChar);
    lcd.createChar(2, pressureChar);
    infoAboutDeviceOutput();
    Serial.println(F("BME280 тест..."));
    //Serial.println("BME280 test...");
    lcd.clear();
    //lcd.print("BME280 test...");
    lcd.print("BME280 тест...");
    unsigned status;
    // налаштування за замовчуванням. Можемо також передати об'єкт бібліотеки Wire як
&Wire2)
    status = bme.begin(BME280_ADDRESS_ALTERNATE);
    if (!status) {
        /*Serial.println("Не можливо знайти робочий сенсор BME280, перевірте підлючення,
адрес, ID - сенсора !");
        Serial.print("SensorID був: 0x"); Serial.println(bme.sensorID(), 16);
        Serial.println("ID 0xFF можливо означає поганий адрес, BMP 180 або BMP 085");
        Serial.println("ID 0x56-0x58 представляє сенсор BMP280");
        Serial.println("ID 0x60 представляє сенсор BME280");
        Serial.println("ID 0x61 представляє сенсор BME680");*/
        Serial.println("Could not find a valid BME280 sensor, check wiring, address, sensor ID!");
        Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
        Serial.println("ID of 0xFF probably means a bad address, a BMP 180 or BMP 085");
        Serial.println("ID of 0x56-0x58 represents a BMP 280");
        Serial.println("ID of 0x60 represents a BME 280");
        Serial.println("ID of 0x61 represents a BME 680");
        lcd.setCursor(0,1);
        lcd.print("couldn't find it");
        while (1);
    }
    Serial.println("пройдено");
    //Serial.println("done");
    lcd.setCursor(0,1);
    lcd.print("done");
    temp = 0.0f; humidity = 0.0f; pressure = 0.0f;
    delayTime = 500;
```

```
    // ініціалізація пінів кнопок як входи:
    pinMode(MENU_BTN_PIN, INPUT);
    pinMode(SELECT_PLUS_BTN_PIN, INPUT);
    pinMode(SELECT_MINUS_BTN_PIN, INPUT);
    pinMode(EXIT_BTN_PIN, INPUT);
    // підключаємо функцію обробки переривання - Attach an interrupt to the ISR vector
    attachInterrupt(digitalPinToInterrupt(interruptPin), ISR_BtnPressed, CHANGE);
    // ініціалізація Wi-Fi модуля і встановлення сервера
    wifiInit();
    client.setServer(mqtt_server[mqtt_server_ind], mqtt_port);
    delay(200);
    lcd.clear();
}

void loop() {
  // зчитуємо температуру, вологість і тиск з сенсора BME280
  temp = bme.readTemperature();
  lcd.setCursor(0,0);
  lcd.write((byte) 0); lcd.print(' ');
  lcd.print(temp,1);
  lcd.print(char(223));
  lcd.print("C ");
  Serial.print("Температура = ");
  //Serial.print("Temperature = ");
  Serial.print(temp);
  Serial.print("\xC2\xB0");
  Serial.println("C");

  humidity = bme.readHumidity();
  lcd.write((byte) 1); lcd.print(' ');
  lcd.print(humidity, 1);
  lcd.print(F("% ")); //lcd.print(F("% RH "));
  Serial.print("Вологість = ");
  //Serial.print("Humidity = ");
  Serial.print(humidity);
  Serial.println("%");

  pressure = bme.readPressure() / 100.0F * 0.7500616827;
  lcd.setCursor(0, 1);
  lcd.write((byte) 2); lcd.print(' ');
  lcd.print(pressure, 1); //(int)pressure);
  lcd.print(" мм.рт.ст");
  //lcd.print(" mmHg");
  Serial.print("Атмосферний тиск = ");
  //Serial.print("Pressure = ");
  Serial.print(pressure);
  Serial.println(" мм.рт.ст.");
  //Serial.println(" mmHg");

  altetude = bme.readAltitude(SEA_LEVEL_PRESSURE_HPA);
  //lcd.print(altetude);
  //lcd.print(" m");
```

```
  Serial.print("Висота = ");
  //Serial.print("Approx. Altitude = ");
  Serial.print(altetude);
  Serial.println(" м");
  //Serial.println(" m");
  Serial.println();

  char jsonBuffer[512];
  size_t jsonMessageSize;

  if (!mqtt_server_ind)
  {
    // {"sensorId":"bme280", "data":[{"temp": 27.83}, {"humidity": 62.05}, {"pressure": 1013.25}]}
    const int capacity = JSON_ARRAY_SIZE(3) + 3*JSON_OBJECT_SIZE(1) +
JSON_OBJECT_SIZE(2);
    //StaticJsonDocument<capacity> doc;
    DynamicJsonDocument doc(capacity);
    doc["sensorId"] = bme.sensorID();
    JsonArray data = doc.createNestedArray("data");
    JsonObject temp_obj = data.createNestedObject();
    temp_obj["temp"] = temp;
    JsonObject humidity_obj = data.createNestedObject();
    humidity_obj["humidity"] = round(humidity*100)/100.0;
    JsonObject pressure_obj = data.createNestedObject();
    pressure_obj["pressure"] = round(pressure*10)/10.0;
    serializeJson(doc, jsonBuffer);
  }
  else {
    // {"temp": 27.83, "humidity": 62.05, "pressure": 1013.25}
    const int capacity = JSON_OBJECT_SIZE(3);
    //StaticJsonDocument<capacity> doc;
    DynamicJsonDocument doc(capacity);
    doc["temp"] = temp;
    doc["humidity"] = round(humidity*100)/100.0;
    doc["pressure"] = round(pressure*10)/10.0;
    serializeJson(doc, jsonBuffer);
  }

  Serial.println("Відсилаємо повідомлення на MQTT-сервер...");
  //Serial.println("Sending message to MQTT broker...");
  Serial.println(jsonBuffer);

  // підключення до Wi-Fi AP і передача даних на MQTT-сервер
  status = WiFi.status();
  if (status != WL_CONNECTED) {
    while (status != WL_CONNECTED) {
      Serial.print(F("Спроба підключитися до WPA SSID: "));
      //Serial.print("Attempting to connect to WPA SSID: ");
      Serial.println(WiFi_ssid[wifi_ind]);
      // підключення до мережі WPA/WPA2
      status = WiFi.begin(WiFi_ssid[wifi_ind], WiFi_passwd[wifi_ind]);
      delay(500);
```

```
    }
    Serial.println(F("Підключено до AP"));
    //Serial.println("Connected to AP");
  }

  if (!client.connected() ) {
    reconnect();
  }
  if ( millis() - lastSend >= 1000 ) { // поновлення і відсилка даних через 1 секунду
     if (client.publish(topic[mqtt_server_ind], jsonBuffer, jsonMessageSize) == true) {
       Serial.println("Повідомлення відіслано успішно");
       //Serial.println("Success sending message");
     } else {
       Serial.println("Помилка відсилання повідомлення");
       //Serial.println("Error sending message");
     }
     lastSend = millis();
   }
   client.loop();

   if (mainMenuInvoke)
     mainMenu();
   delay(delayTime);
}
```