НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

Касаткін Д.Ю., к. пед.н., доц.

Підпис ПІБ, вчене звання і ступінь

р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: Створення комп’ютерної системи спостереження периметру фермерського приміщення

Спеціальність 123 «Комп’ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доцент _____ Євгеній НІКІТЕНКО

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник випускної бакалаврської роботи

к.пед.н., доцент _____ Дмитро КАСАТКІН

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав

(підпис) (ПІБ студента)

Київ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАНН УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки

Касаткін Д.Ю., к.пед.н., доц. /
підпис ПІБ, вчене звання і ступінь
р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТА

Морозюк Роман Віталійович
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки) 123 Комп'ютерна інженерія
Тема випускної бакалаврської роботи Створення комп'ютерної системи спостереження периметру фермерського приміщення
керівник проекту (роботи) Касаткін Д.Ю., к.пед.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджено наказом ректора НУБіП України від «16» грудня 2024 р. № 2250 «С»
Термін подання завершеної роботи на кафедру 28.05.2025
(рік, місяць, число)

Вихідні дані до випускної бакалаврської роботи

Перелік питань, які потрібно розробити: Аналіз вимог до комп'ютерної системи спостереження, аналіз предметної області, проєктування, реалізація, тестування системи

Перелік графічних документів (за потреби)

Дата видачі завдання "17" грудня 2024 р
Керівник випускної бакалаврської роботи / Касаткін Д.Ю.
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання / Морозюк Р.В.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз вимог до системи	07.03.2025 р.	Виконано
2	Проєктування системи	21.03.2025 р.	Виконано
3	Реалізація системи	12.04.2025 р.	Виконано
4	Тестування розробленої системи	27.04.2025 р.	Виконано
5	Оформлення пояснювальної записки	07.05.2025 р.	Виконано
6	Оформлення графічного матеріалу	15.05.2025 р.	Виконано

Студент_____ / Морозюк Р.В. /

(підпис) (ініціали та прізвище)

Керівник проекту (роботи)_____ / Касаткін Д.Ю./

(підпис) (ініціали та прізвище)

ЗМІСТ

ВСТУП

Актуальність роботи. Тема "Комп’ютерна система моніторингу навколишнього середовища на базі Arduino" є вкрай актуальною в сучасних умовах, коли питання екологічного контролю та автоматизації процесів управління кліматичними умовами набувають все більшої важливості. З розвитком технологій "розумних" будинків, системи моніторингу навколишнього середовища стають ключовими елементами для підвищення енергоефективності житлових приміщень.

Розробка та впровадження комп'ютерної системи моніторингу навколишнього середовища на базі Arduino передбачає створення надійної метеостанції для збору даних про кліматичні умови в житлових приміщеннях. Така система буде використовувати датчики для вимірювання температури та вологості, а також можливість інтеграції з системою опалення через Wi-Fi модулі для автоматичного керування котлом на основі отриманих даних. Ця система забезпечить оптимальне регулювання температури в приміщенні, що сприятиме підвищенню комфорту та зниженню витрат на енергію.

Отже, кваліфікаційна робота на тему "Комп’ютерна система моніторингу навколишнього середовища на базі Arduino" є актуальною і корисною, оскільки її результати дозволять розробити ефективні методи автоматизації контролю кліматичних умов у приміщеннях. Впровадження подібних систем стане важливим кроком на шляху до підвищення енергоефективності житлових будівель, а також сприятиме розвитку "розумних" екосистем, що знижують навантаження на природні ресурси.

Крім того, дана робота сприятиме подальшому розвитку технологій у галузі інтернету речей (IoT), що є актуальним в контексті глобальних тенденцій автоматизації побутових та промислових процесів.

Метою роботи є розроблення комп’ютерної системи моніторингу навколишнього середовища на базі Arduino, яка забезпечить збір, обробку та аналіз даних про екологічні параметри в реальному часі.

Отже, мета дипломної роботи полягає в тому, щоб дослідити та реалізувати ефективні методи моніторингу навколишнього середовища, використовуючи платформи Arduino. Дипломна робота має на меті розробку системи, яка дозволить відстежувати важливі екологічні показники, такі як температура, вологість, рівень забруднення повітря та інші параметри, що впливають на здоров'я людини та стан екосистеми. Результатом дослідження повинно стати інтерактивне програмне забезпечення, яке дозволяє користувачам візуалізувати дані та отримувати рекомендації щодо покращення екологічної ситуації.

Для досягнення поставленої мети в роботі вирішені такі завдання:

1. Аналіз поточних методів моніторингу навколишнього середовища та їх ефективності.
2. Вибір та інтеграція відповідних датчиків для вимірювання екологічних параметрів.
3. Розробка системи збору та обробки даних за допомогою платформи Arduino.
4. Розробка програмного забезпечення для візуалізації отриманих даних та їх аналізу.
5. Проведення експериментальної перевірки функціонування системи та її точності.
6. Розробка рекомендацій щодо покращення екологічної ситуації на основі отриманих даних.

Результати дослідження дипломної роботи можуть бути корисні для організацій та установ, які займаються моніторингом навколишнього середовища, а також для наукових досліджень у галузі екології. Система дозволить отримувати актуальну інформацію про екологічні умови та вживати заходів для їх покращення.

Об’єкт дослідження – процеси комп’ютерної системи моніторингу навколишнього середовища на базі Arduino. Дослідження охоплює аналіз різних аспектів системи, включаючи вибір датчиків, архітектуру системи, алгоритми збору та обробки даних, а також способи

візуалізації інформації. Особлива увага приділяється методам аналізу екологічних параметрів, які впливають на здоров’я людини та стан навколишнього середовища.

Кваліфікаційна робота зосереджується на вивченні конкретної системи моніторингу навколишнього середовища, але її результати можуть бути застосовані для розуміння загальних принципів та методів ефективного моніторингу екологічних умов у різних умовах

Предмет дослідження – розроблення комп’ютерної системи моніторингу навколишнього середовища на базі Arduino, з урахуванням різних аспектів збору та обробки даних, таких як вибір датчиків, алгоритми обробки інформації, візуалізація результатів моніторингу, а також інтеграція системи з іншими платформами. Дослідження також охоплює використання методів аналізу екологічних даних, забезпечення точності вимірювань, а також способів покращення системи на основі отриманих результатів.

Методи дослідження. Аналіз документів та літературних джерел, експертні оцінки, тестування та аудит системи моніторингу, моделювання екологічних процесів, а також практичні експерименти для перевірки точності та надійності отриманих даних.

Інформаційною базою досліджень є:

Науково-технічна література з тематики моніторингу навколишнього середовища, технологій Arduino, обробки даних та екологічної безпеки.

Законодавчі акти, що регулюють питання екологічного моніторингу та охорони навколишнього середовища.

Документація від виробників датчиків та обладнання для збору екологічних даних.

Стандарти та рекомендації організацій, які встановлюють вимоги до точності та надійності систем моніторингу.

Результати досліджень та проекти з розробки та впровадження систем моніторингу навколишнього середовища в інших організаціях.

Експертні оцінки та поради фахівців з екологічного моніторингу та системи автоматизації.

Дані та статистика з практики застосування сучасних систем моніторингу навколишнього середовища на підприємствах різних галузей.

Практичне значення отриманих результатів. Отримані результати дослідження можуть мати практичне значення, зокрема, щодо вдосконалення системи моніторингу навколишнього середовища, забезпечення точності та надійності збору даних, зменшення витрат на експлуатацію та обслуговування системи, а також покращення реакції на зміни екологічних умов. Розроблена система може допомогти підприємствам ефективно відстежувати стан навколишнього середовища, вживати заходів для його покращення та забезпечувати екологічну безпеку.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ІСНУЮЧИХ КОМП'ЮТЕРНИХ СИСТЕМ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА НА БАЗІ ARDUINO

1.1. Структура комп'ютерної системи моніторингу навколишнього середовища на базі Arduino

Комп'ютерна система моніторингу навколишнього середовища на базі Arduino – це інтегрована система, яка забезпечує збір, обробку та аналіз даних про екологічні параметри, такі як температура, вологість, рівень забруднення та інші важливі показники. Ця система повинна бути надійною та точною, щоб забезпечити достовірність інформації про навколишнє середовище.

Основна мета системи моніторингу – забезпечити своєчасний збір та аналіз екологічних даних, що дозволяє вживати необхідні заходи для покращення стану навколишнього середовища та забезпечення здоров'я людей. Завдяки швидкому обміну інформацією між датчиками та центральним процесором, система підвищує ефективність управлінських рішень.

Крім того, система моніторингу навколишнього середовища дозволяє зберігати та аналізувати дані, що надходять, за допомогою спеціалізованого програмного забезпечення, що включає в себе алгоритми обробки та візуалізації інформації. Це дозволяє користувачам отримувати зручний доступ до аналітичних даних, прогнозувати зміни в екологічних умовах і вживати необхідні заходи.

Основні складові системи моніторингу включають в себе різноманітні датчики (температури, вологості, якості повітря), мікроконтролери (таких як Arduino), модулі зв'язку (наприклад, Wi-Fi або Bluetooth) для передачі даних, а також програмне забезпечення для обробки та аналізу отриманих даних. Це може також включати бази даних для зберігання даних та системи візуалізації для представлення результатів моніторингу.

У сучасному екологічному контексті система моніторингу навколишнього середовища є важливою для забезпечення сталого розвитку підприємств, що задіяні у виробництві або наданні послуг, які можуть впливати на екологію.

Комплексний підхід до моніторингу екологічних параметрів допомагає у виявленні проблем на ранніх стадіях, що є критично важливим для ефективного управління ресурсами та зменшення негативного впливу на довкілля.

Окрім технічних рішень, важливою частиною системи є підготовка користувачів до роботи з нею. Вони повинні бути обізнані про екологічні проблеми та методи, які допоможуть виявляти та усувати негативні чинники.

Крім того, система повинна бути здатною швидко реагувати на зміни в навколишньому середовищі, що може включати автоматизовані сповіщення про перевищення допустимих значень параметрів, а також можливість віддаленого моніторингу[3].

З розширенням використання технологій Internet of Things (IoT) система моніторингу навколишнього середовища на базі Arduino може стати важливим елементом у глобальній інфраструктурі, що забезпечує доступ до екологічних даних у реальному часі з будь-якої точки світу. У сучасному світі, де питання охорони навколишнього середовища стає дедалі актуальнішим, розробка та впровадження таких систем є важливим кроком до сталого розвитку[1].

1.2. Огляд існуючих комп'ютерних систем моніторингу навколишнього середовища на базі Arduino

У цьому підрозділі буде проведено детальний аналіз існуючих комп'ютерних систем моніторингу навколишнього середовища, що базуються на платформі Arduino, з акцентом на їхню структуру, функціональні можливості та практичні застосування. Аналіз включатиме порівняння різних проектів, їх переваги та недоліки, а також дослідження впливу таких систем на управління ресурсами та охорону навколишнього середовища. Зокрема, буде розглянуто, як ці технології

сприяють підвищенню ефективності збору даних та прийняття рішень на основі отриманих результатів[4].

1. Типи систем моніторингу

1. **Стаціонарні системи моніторингу**: Стаціонарні системи, такі як Arduino Weather Station, використовуються для збору даних про погоду в конкретних місцях. Ці системи зазвичай оснащені такими датчиками, як DHT11 або BME280, які вимірюють температуру, вологість та атмосферний тиск. Вони можуть передавати зібрані дані через Wi-Fi або GSM для подальшого аналізу та зберігання в базах даних. Наприклад, Weather Station Project може передавати дані в реальному часі на веб-інтерфейс, де користувачі можуть переглядати інформацію про погодні умови у зручному форматі.

2. **Мобільні системи моніторингу**: Мобільні системи, які використовують модулі GPS, дозволяють здійснювати моніторинг якості повітря в різних точках міста. Проекти, такі як Air Quality Monitoring System, можуть включати датчики MQ-135 для виявлення рівня забруднення повітря. Такі системи часто обладнані батареями, що дозволяє їм працювати автономно і переміщатися в межах визначеної території.

2. Складові частини системи

Датчики є основними компонентами системи моніторингу, які дозволяють збирати точну інформацію про навколишнє середовище. У залежності від типу контрольованих параметрів, застосовуються різні типи датчиків. Ось деякі з основних типів датчиків:

- Датчики температури і вологості:
  - DHT11 і DHT22 – Це одні з найбільш поширених датчиків, що вимірюють температуру та вологість повітря. DHT11 є базовим варіантом із середньою точністю, але для застосувань, що потребують більшої точності, краще використовувати DHT22. Ці датчики можуть забезпечувати дані про кліматичні умови, що дозволяє оцінювати загальний комфорт в приміщенні або на вулиці.

- BME280 – Цей датчик здатний вимірювати не лише температуру та вологість, але й атмосферний тиск. Це дозволяє розширити спектр вимірювань для аналізу погодних умов та створення локальних метеорологічних прогнозів. BME280 має високу точність та компактний розмір, що робить його зручним для інтеграції в невеликі пристрої.

- Датчики якості повітря:
  - MQ-135 – Датчик якості повітря, який чутливий до ряду газів, таких як аміак, бензол та дим. Це дозволяє оцінювати рівень забруднення в закритих приміщеннях або поблизу джерел викидів.
  - MQ-7 – Датчик, спеціально призначений для вимірювання концентрації чадного газу (CO), що особливо корисно в системах моніторингу безпеки, наприклад, для попередження про можливі витоки в приміщеннях, де можуть бути потенційні джерела чадного газу.
- Датчики ультрафіолетового випромінювання:
  - VEML6075 – Цей датчик вимірює рівень ультрафіолетового випромінювання типу UVA та UVB. Він дозволяє оцінювати вплив сонячної радіації на навколишнє середовище, що може бути важливим для моніторингу стану рослин, а також для попередження про небезпеку опромінення для людей у відкритих зонах.

Датчики в цій системі не лише зчитують показники, але й допомагають виявляти аномальні умови, що можуть потребувати негайного реагування, наприклад, надмірне забруднення повітря або критично високий рівень вологості.

2.2. Модулі зв'язку

Для ефективного обміну інформацією між різними компонентами системи та зовнішніми серверами використовуються модулі зв'язку, що забезпечують безперебійну передачу даних.

- Wi-Fi модулі:
  - ESP8266 і ESP32 – Це найпопулярніші модулі для бездротової передачі даних, які дозволяють підключатися до локальних мереж та Інтернету. ESP8266

використовується для базових завдань, тоді як ESP32 має розширені можливості, включаючи більшу кількість портів та вбудовану підтримку Bluetooth. Використання Wi-Fi дозволяє налаштувати систему для передачі даних у реальному часі, що є важливим для оперативного реагування на зміни умов.

- Wi-Fi модулі особливо зручні у випадках, коли існує стабільне підключення до Інтернету. Вони забезпечують віддалений доступ до даних через веб-інтерфейси або мобільні додатки, дозволяючи користувачам швидко переглядати актуальні дані.

- GSM модулі:
  - SIM800 і SIM900 – Ці модулі призначені для систем, що працюють в умовах, де немає Wi-Fi або провідного Інтернету. Вони дозволяють підключитися до мобільних мереж та передавати дані через GPRS, що особливо корисно для мобільних систем або в сільській місцевості. GSM-модулі можуть використовуватися для надсилання сповіщень у вигляді SMS, коли показники перевищують встановлені межі, що допомагає запобігати аварійним ситуаціям.
  - Така система є автономною та забезпечує надійний зв’язок незалежно від розташування, що робить її ідеальною для моніторингу умов в віддалених районах.

2.3. Програмне забезпечення

Для обробки та візуалізації даних, зібраних датчиками, використовуються спеціалізовані програмні платформи, що спрощують процес аналізу та роблять систему доступною для кінцевих користувачів.

- ThingSpeak:
  - ThingSpeak є однією з популярних платформ для зберігання даних IoT. Вона дозволяє зберігати дані на хмарних серверах, а також надає інструменти для їх візуалізації. Це дозволяє користувачам створювати графіки, що відображають зміни умов з часом, і встановлювати порогові значення для сигналів тривоги.

- ThingSpeak також підтримує аналітику та моделі машинного навчання для прогнозування можливих змін на основі зібраних даних, що допомагає користувачам здійснювати проактивні дії для запобігання потенційним проблемам.

- Blynk:
  - Blynk – це платформа, яка надає мобільний додаток для відображення даних у режимі реального часу. Її простий інтерфейс дозволяє користувачам без спеціальних навичок налаштовувати віджети для відображення показників від датчиків. Мобільні сповіщення від Blynk допомагають відразу реагувати на зміни в навколишньому середовищі, роблячи систему ще більш зручною та ефективною.
  - Blynk надає гнучкі можливості для інтеграції з різними пристроями на базі Arduino, завдяки чому користувачі можуть керувати параметрами роботи системи та оперативно змінювати налаштування через інтуїтивний інтерфейс.

Таким чином, всі компоненти системи моніторингу працюють взаємопов’язано, забезпечуючи збирання, передачу та обробку даних, що дозволяє користувачам отримувати повну картину про стан навколишнього середовища в режимі реального часу та реагувати на можливі ризики завчасно.

3. Методи збору та обробки даних

Комп’ютерна система моніторингу навколишнього середовища на базі Arduino реалізує багатоетапний процес збору та обробки даних, який дозволяє отримувати точні та актуальні показники про стан середовища в режимі реального часу. Етапи цього процесу охоплюють збирання, обробку, фільтрацію, зберігання та передачу даних з урахуванням вимог до точності та швидкодії[3].

1. Збір даних

На першому етапі система використовує різноманітні датчики для збору інформації про навколишнє середовище. Вибір частоти збору даних залежить від цілей моніторингу та типу параметрів, що вимірюються. Для точності вимірювань та уникнення перешкод встановлюються оптимальні значення інтервалів між зчитуванням даних для кожного типу датчиків:

- Частота збору даних: Система може бути налаштована на збір даних з інтервалом від 1 до 15 секунд для параметрів, що швидко змінюються, наприклад, температура або вологість. У випадку повільніших процесів, як-то зміни рівня ультрафіолетового випромінювання, частота вимірювання може бути більш рідкою — від кількох хвилин до годин.
- Захист від шумів: Для забезпечення точності даних при зборі вимірювань Arduino може використовувати функції усереднення показників. Наприклад, якщо датчик температури зчитує значення кожні 5 секунд, система може обчислити середнє значення для 12 вимірювань (за хвилину) та використовувати його як стабільний показник для подальшої обробки. Це дозволяє мінімізувати випадкові відхилення через перешкоди або короткочасні коливання.

2. Обробка даних

Другий етап — це обробка даних мікроконтролером Arduino, який виконує початкові обчислення і обробку, перш ніж передавати дані на сервер. Обробка даних включає кілька важливих підетапів, що забезпечують їх достовірність та готовність до передачі.

- Алгоритми фільтрації: Для видалення шумів та випадкових помилок Arduino використовує алгоритми цифрової фільтрації, такі як рухоме середнє або експоненційне згладжування. Це дозволяє отримати більш точні та стабільні показники.
- Корекція похибок: Деякі датчики можуть давати невеликі похибки в результатах через зовнішні фактори (наприклад, температурний датчик може спотворювати дані через перегрів самого мікроконтролера). Для мінімізації цих похибок система може здійснювати корекцію результатів на основі попередньо встановлених коефіцієнтів калібрування.
- Форматування даних: Отримані дані можуть бути приведені до зручного для подальшої передачі формату. Наприклад, температура може конвертуватися у потрібні одиниці (градуси Цельсія або Фаренгейта), залежно

від специфікацій системи та потреб користувача. Форматування також дозволяє забезпечити зручність роботи з даними на наступних етапах.

3. Зберігання даних

Після обробки дані можуть зберігатися як на локальному носії, так і в хмарному сховищі. Локальне зберігання дозволяє використовувати дані для аналізу в умовах обмеженого зв'язку, а хмарне сховище забезпечує доступність даних в будь-який час для віддаленого перегляду.

- Локальне зберігання на SD-карті: У випадках, коли відсутній доступ до мережі, дані можуть зберігатися на SD-карті, підключеній до Arduino. Це дозволяє накопичувати великі обсяги інформації, які можна аналізувати пізніше, коли з'явиться можливість доступу до мережі.
- Хмарне зберігання: За наявності інтернет-з'єднання дані можуть передаватися на хмарний сервер (наприклад, на платформи ThingSpeak або Google Cloud). Це дає змогу забезпечити тривале зберігання даних і доступ до них з різних пристроїв, а також можливість створювати резервні копії для захисту від втрати інформації.

4. Передача даних

На цьому етапі оброблені дані передаються на сервер або хмарне сховище для подальшого аналізу, візуалізації та зберігання. Передача може здійснюватися кількома способами, залежно від умов роботи системи та вимог до безпеки.

- Передача через Wi-Fi: Якщо система знаходиться в зоні доступу до Wi-Fi, модуль ESP8266 або ESP32 може забезпечити передачу даних через мережу Інтернет. Це дозволяє оперативно передавати дані на сервер для їх обробки, забезпечуючи віддалений доступ до актуальних показників.
- Передача через GSM: Для мобільних систем або для роботи в умовах, де немає доступу до Wi-Fi, використовуються GSM-модулі (SIM800, SIM900), які забезпечують передачу даних через стільниковий зв'язок. Це дозволяє здійснювати моніторинг навіть у віддалених районах, що може бути критично важливим для екологічних досліджень у важкодоступних місцях.

- Захист даних під час передачі: Для запобігання перехопленню даних використовуються методи шифрування, такі як SSL/TLS, особливо для передачі конфіденційної інформації. Це підвищує надійність та безпеку системи.

5. Візуалізація та аналіз даних

Останнім кроком у цьому процесі є візуалізація зібраних даних на основі аналітичних платформ. Це дозволяє користувачам легко спостерігати за змінами параметрів навколишнього середовища та швидко реагувати на відхилення.

- Візуалізація на графіках: Зібрані дані автоматично будуються у вигляді графіків, що показують зміни параметрів у реальному часі. Наприклад, температурний графік дозволяє відстежувати коливання протягом доби, а графік вологості показує рівень вологості у різні періоди дня.
- Налаштування порогових значень: Система дозволяє встановлювати порогові значення, при досягненні яких активуються сповіщення. Наприклад, якщо рівень CO2 перевищує допустимий, користувач отримує сповіщення на смартфон чи комп’ютер, що дає змогу оперативно реагувати.
- Прогнозування та аналітика: Завдяки платформам, таким як ThingSpeak, можна застосувати моделі машинного навчання для прогнозування можливих змін параметрів на основі попередніх даних. Це дає можливість передбачити екологічні ризики, наприклад, прогнозування підвищення температури в літній період або зниження вологості в осінньо-зимовий сезон.

6. Додаткові алгоритми для покращення роботи системи

Окрім стандартних методів збору та обробки даних, система може використовувати додаткові алгоритми для забезпечення точності, надійності та ефективності роботи.

- Система самообслуговування та калібрування: У залежності від умов експлуатації, система може бути налаштована на автоматичне калібрування сенсорів, щоб уникнути похибок. Це досягається за допомогою

калібрувальних таблиць, які враховують зовнішні фактори, що впливають на точність показників.

- Віддалене налаштування параметрів: Дистанційне управління системою дозволяє змінювати налаштування частоти зчитувань, порогові значення для сповіщень та інші параметри без фізичного втручання. Це забезпечує гнучкість у використанні та полегшує управління системою.

Таким чином, методи збору та обробки даних у системах моніторингу на базі Arduino забезпечують високий рівень точності та надійності, що дозволяє оперативно отримувати інформацію про навколишнє середовище для прийняття ефективних рішень[2].

4. Приклади реалізації проектів

1. **Arduino-based Air Quality Monitor**: Ця система є чудовим прикладом використання Arduino для моніторингу екологічних параметрів. Система оснащена датчиками якості повітря, такими як MQ-135 для вимірювання рівня вуглекислого газу, MQ-7 для контролю чадного газу, а також іншими датчиками для виявлення запахів та летючих органічних сполук (ЛОС). Arduino збирає дані з цих датчиків і передає їх через Wi-Fi або GSM модуль на веб-інтерфейс, де користувачі можуть переглядати інформацію в режимі реального часу. Крім цього, система може надсилати сповіщення на смартфон або електронну пошту у випадку, якщо рівень забруднення досягає критичних значень, що дозволяє оперативно реагувати на зміну якості повітря. Система також може бути інтегрована з додатковими компонентами, як-от вентиляційні пристрої, які автоматично активуються у відповідь на високу концентрацію забруднюючих речовин.

2. **Arduino Weather Station**: Це комплексний проект, призначений для моніторингу погодних умов в реальному часі. В системі використовуються датчики для вимірювання температури, вологості, атмосферного тиску, а також анемометр для визначення швидкості вітру. Станція може бути доповнена ультрафіолетовим сенсором для визначення рівня сонячної радіації, що корисно

для оцінки безпеки перебування на сонці. Всі зібрані дані передаються на сервер або інтернет-платформу, наприклад, ThingSpeak або Blynk, де відображаються у вигляді графіків. Це дозволяє користувачам аналізувати зміни погодних умов протягом тривалого часу. Для зручності, система може надсилати щоденні зведення або сповіщення про різке зниження температури, що може бути корисним для сільського господарства або управління побутовими системами опалення.

3. **Система моніторингу вологості ґрунту для сільського господарства**: Ця система використовує Arduino для моніторингу вологості ґрунту з використанням спеціалізованих датчиків, таких як YL-69 або FC-28. Датчики зчитують рівень вологості, а мікроконтролер Arduino обробляє ці дані та, за потреби, надсилає сигнали для автоматичного зрошення. Система може бути інтегрована з Wi-Fi модулем, що дозволяє передавати дані на смартфон або ПК, де користувачі можуть дистанційно контролювати стан ґрунту. Це особливо корисно для фермерів, які можуть своєчасно вживати заходів для підтримки оптимального рівня зволоження, що сприяє підвищенню врожайності[3].

4. **Домашня система моніторингу енергоспоживання**: За допомогою Arduino та датчиків струму, таких як ACS712, можна створити систему, яка вимірює енергоспоживання різних електричних пристроїв у будинку. Arduino зчитує поточне енергоспоживання і передає ці дані на сервер або додаток для моніторингу. Це дозволяє користувачам аналізувати споживання електроенергії, оптимізувати використання приладів і знижувати витрати на електроенергію. Система може також мати функцію сповіщення у разі перевищення встановленого ліміту споживання, що дає змогу уникнути перевантажень в мережі або поломок.

5. **Контроль освітлення та температури в теплицях**: Arduino-based система для управління кліматом у теплиці може автоматично підтримувати необхідні умови для рослин. Вона використовує датчики температури і вологості, а також датчики освітленості для аналізу поточних параметрів клімату. Arduino контролює інтенсивність освітлення, вентиляцію та полив, базуючись на зібраних

даних. Завдяки з'єднанню з мобільним додатком або веб-інтерфейсом, користувач може спостерігати за умовами у теплиці, а також дистанційно налаштовувати параметри освітлення чи температуру[2].

5. Виклики та перспективи

1. **Надійність систем**: Одним із основних викликів при впровадженні систем моніторингу є забезпечення стабільної та надійної роботи, особливо у віддалених місцях або регіонах із нестабільним зв’язком. Система повинна мати можливість продовжувати збір даних навіть при відсутності інтернет-з’єднання, а накопичені дані повинні автоматично передаватися на сервер одразу після відновлення зв’язку. Важливим аспектом є також енергозалежність системи, оскільки для довготривалого функціонування необхідно впроваджувати енергоефективні технології, використання сонячних панелей або інших автономних джерел живлення, що забезпечить тривалу роботу без підключення до електромережі.
2. **Безпека даних**: Збереження конфіденційності та цілісності даних стає дедалі важливішим, оскільки системи моніторингу дедалі частіше стають об'єктами кіберзагроз. Зловмисники можуть спробувати змінити або викрасти зібрану інформацію, що може призвести до негативних наслідків, особливо в системах критичної інфраструктури. Для захисту даних необхідно впроваджувати сучасні протоколи шифрування, наприклад, SSL або TLS, що забезпечать безпечне передавання даних. Також доцільним є впровадження багаторівневої системи аутентифікації та авторизації, що мінімізує ризик несанкціонованого доступу до даних та управління системою[5].
3. **Перспективи розвитку**: Розвиток Інтернету речей (IoT) відкриває значні можливості для вдосконалення систем моніторингу навколишнього середовища. Впровадження штучного інтелекту (ШІ) та технологій машинного навчання дозволяє автоматизувати аналіз зібраних даних. Зокрема, система може

самостійно виявляти аномалії, будувати прогнози на основі історичних даних та виконувати предиктивний аналіз, що дозволяє передбачати небезпечні ситуації. Такі функції допомагають не тільки спостерігати за станом середовища, але й здійснювати превентивні заходи, які значно підвищують рівень безпеки та надійності.

4. **Масштабованість і гнучкість**: У зв'язку зі зростанням потреб у моніторингу та аналізі великих обсягів даних, важливим стає питання масштабованості системи. Сучасні системи моніторингу повинні бути гнучкими, з можливістю інтеграції нових датчиків, розширення функціоналу та додавання нових модулів для різних параметрів моніторингу. Це особливо важливо в умовах, коли виникає необхідність відстежувати нові екологічні фактори або інтегрувати додаткові пристрої для покращення якості даних[5].

5. **Екологічна відповідальність**: Зростання екологічної свідомості серед споживачів стимулює розвиток систем моніторингу з акцентом на енергоефективність та мінімізацію відходів. Інтеграція таких систем у громадські місця або промислові об'єкти також повинна відповідати вимогам щодо зниження впливу на довкілля.

1.3. Аналіз існуючих сенсорних технологій для моніторингу навколишнього середовища

У цьому підрозділі буде проведено детальний аналіз різноманітних сенсорних технологій, які використовуються в системах моніторингу навколишнього середовища на базі Arduino. Сенсори є ключовими компонентами в таких системах, оскільки вони забезпечують необхідні дані для оцінки стану навколишнього середовища. Розглянемо основні категорії сенсорів, їх характеристики та практичні

застосування, а також порівняємо їх ефективність у контексті різних завдань моніторингу[6].

1.3.1. Сенсори температури та вологості

Сенсори температури та вологості, такі як DHT11 та DHT22, широко використовуються в системах моніторингу для контролю мікроклімату.

- **DHT11**: Цей сенсор має обмежену точність і діапазон, проте він доступний за низькою ціною і підходить для базових застосувань. DHT11 вимірює температуру в діапазоні від 0 до 50 °C з точністю ±2 °C та відносну вологість від 20% до 90% з точністю ±5%. Це робить його ідеальним для домашніх проектів та невеликих установок, де не потрібно високої точності.
- **DHT22**: На відміну від DHT11, DHT22 пропонує вищу точність і більший діапазон вимірювань. Він вимірює температуру від -40 до 80 °C з точністю ±0.5 °C та вологість від 0% до 100% з точністю ±2-5%. Завдяки цьому, DHT22 є більш оптимальним варіантом для складніших застосувань, таких як агрономічні дослідження або контролювання умов у теплицях.

Обидва сенсори можуть бути інтегровані з Arduino для автоматизованого збору даних, що дозволяє здійснювати моніторинг у реальному часі. Зібрані дані можуть бути візуалізовані на графіках або збережені для подальшого аналізу, що робить їх надзвичайно корисними для вчених та дослідників[7].

1.3.2. Сенсори якості повітря

Сенсори якості повітря, такі як MQ-2 і MQ-135, використовуються для вимірювання концентрацій різних забруднювачів у повітрі, включаючи чадний газ, метан та інші летючі органічні сполуки.

- **MQ-2**: Цей сенсор є універсальним і здатний виявляти різні гази, такі як пропан, метан і чадний газ. MQ-2 має чутливість до діапазону від 300 до 10,000 ppm (частин на мільйон), що робить його корисним для моніторингу витоків газу у побуті чи на виробництвах. Його

використання може запобігти небезпечним ситуаціям, пов'язаним з вибухонебезпечними газами.

- **MQ-135**: Сенсор, який спеціалізується на визначенні якості повітря, може вимірювати концентрації таких забруднювачів, як бензол, аміак і CO2. MQ-135 забезпечує вимірювання в широкому діапазоні, що дозволяє отримувати точні дані про загальний стан повітря. Це робить його незамінним для моніторингу забруднення повітря в містах, а також для контролю якості повітря в закритих приміщеннях, таких як офіси та лабораторії[9].

Системи моніторингу з використанням цих сенсорів можуть забезпечити своєчасне попередження про можливі проблеми зі здоров'ям, пов'язані із забрудненням повітря, що особливо важливо в умовах сучасного урбанізованого середовища.

1.3.3. Сенсори рівня води та опадів

Сенсори рівня води, такі як ультразвукові сенсори HC-SR04 або резистивні датчики, застосовуються для вимірювання рівня води у водоймах, резервуарах та системах зрошення.

- **HC-SR04**: Цей ультразвуковий сенсор дозволяє вимірювати відстань до поверхні води. Він має максимальний діапазон вимірювань до 4 метрів з точністю до ±3 мм. Це робить його ідеальним для моніторингу рівня води у водоймах або для автоматичного контролю в системах зрошення, що сприяє економії води.
- **Резистивні датчики**: Використовують електричний опір для визначення рівня води. Вони прості в установці і можуть бути використані в системах автоматичного зрошення для контролю потреб рослин. Ці датчики забезпечують точні дані, які допомагають фермерам оптимізувати витрати води та підвищити врожайність.

Системи моніторингу з використанням цих сенсорів можуть також використовуватися для прогнозування повеней, забезпечуючи своєчасне реагування на підвищення рівня води у ріках чи водоймах[8].

1.3.4. Сенсори світла

Сенсори світла, такі як фоторезистори або сенсори LDR (Light Dependent Resistor), використовуються для вимірювання рівня освітленості в середовищі.

- **Фоторезистори (LDR)**: Ці компоненти змінюють свій електричний опір залежно від інтенсивності світла. Вони ідеально підходять для використання в системах автоматизації освітлення, де світло може бути включено або вимкнено залежно від рівня природного освітлення. Наприклад, у розумних будинках фоторезистори можуть використовуватися для автоматичного регулювання освітлення, знижуючи споживання електроенергії.
- **Сенсори RGB**: Ці сенсори можуть вимірювати інтенсивність світла в різних спектрах (червоному, зеленому, синьому), що дозволяє здійснювати детальніший аналіз освітленості. Вони можуть бути використані для виявлення зміни кольору та інтенсивності світла, що важливо в агрономії та ботаніці.

Ці сенсори можуть також бути використані в системах для моніторингу фотосинтетичних процесів у рослинництві, що дозволяє оптимізувати умови вирощування[11].

1.3.5. Інтеграція сенсорів з платформою Arduino

Важливою частиною будь-якої системи моніторингу є спосіб інтеграції сенсорів з платформою Arduino. Для цього використовуються різноманітні бібліотеки, які спрощують процес збору та обробки даних. Наприклад, бібліотеки для роботи з DHT-сенсорами або MQ-сенсорами дозволяють легко отримувати дані та передавати їх на зовнішні платформи для аналізу[12].

Завдяки простоті підключення та налаштування Arduino, користувачі можуть швидко реалізувати свої проекти та отримувати результати без значних витрат часу на програмування. Багато з цих сенсорів підтримують стандартні протоколи зв'язку, такі як I2C або SPI, що ще більше спрощує інтеграцію з іншими пристроями та системами[13].

Крім того, зібрані дані можуть бути збережені у базах даних для подальшого аналізу або навіть передані на хмарні платформи для створення системи моніторингу з доступом у реальному часі. Такі рішення дозволяють користувачам здійснювати моніторинг із будь-якої точки світу, що робить їх надзвичайно гнучкими та ефективними[14].

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз комп'ютерних систем моніторингу навколишнього середовища на базі Arduino, зосереджуючись на їх структурі, огляді існуючих рішень та сенсорних технологіях.

У підрозділі 1.1 було розглянуто основні компоненти, що формують комп'ютерну систему моніторингу, включаючи апаратні та програмні складові. Визначено, що система складається з сенсорів, мікроконтролерів, модуля зв'язку та програмного забезпечення для обробки даних. Основна мета цієї системи полягає в забезпеченні точного та своєчасного збору даних про стан навколишнього середовища, що дозволяє здійснювати моніторинг і реагувати на зміни в умовах.

У підрозділі 1.2 було проведено огляд існуючих комп'ютерних систем моніторингу, акцентуючи увагу на їхніх функціональних можливостях та перевагах. Виявлено, що системи, які базуються на платформі Arduino, відрізняються простотою реалізації, доступністю компонентів та широкими можливостями налаштування.

У підрозділі 1.3 був здійснений аналіз сенсорних технологій, що використовуються для збору даних про навколишнє середовище. Виявлено, що існує широкий вибір сенсорів для вимірювання температури, вологості, якості повітря, рівня води та освітленості, кожен з яких має свої переваги та специфікації. Інтеграція цих сенсорів з платформою Arduino забезпечує гнучкість у створенні різноманітних систем моніторингу, які можуть бути адаптовані під конкретні потреби.

Отже, розглянуті теми підтверджують актуальність і доцільність використання комп'ютерних систем моніторингу навколишнього середовища на базі Arduino. Вони не тільки сприяють підвищенню ефективності моніторингу, але й забезпечують можливість реалізації нових підходів до управління ресурсами в різних сферах.

РОЗДІЛ 2. ВИБІР ІНСТРУМЕНТІВ ДЛЯ КОМП'ЮТЕРНИХ СИСТЕМ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА НА БАЗІ ARDUINO

2.1 Етапи побудови системи моніторингу навколишнього середовища на базі Arduino

Побудова комп'ютерної системи моніторингу навколишнього середовища на базі Arduino є складним і багатоступеневим процесом, що вимагає детального планування та аналізу. Це допомагає забезпечити ефективне та надійне функціонування системи. Основні етапи цієї побудови включають:

1. Аналіз вимог до системи: Першим кроком у створенні системи є ретельний аналіз вимог користувачів. Цей етап передбачає детальне визначення типів даних, які будуть моніторитися (температура, вологість, забруднення повітря, рівень шуму тощо), а також кількість сенсорів, які необхідні для збору цих даних. Кожен сенсор повинен бути обраний з урахуванням його специфікацій, точності та умов, в яких він буде експлуатуватися. Крім того, необхідно врахувати, які саме параметри будуть критично важливими для підприємства, щоб визначити пріоритети в зборі та обробці даних.

2. Розробка архітектури системи: На основі проведеного аналізу розробляється архітектура системи, що включає визначення типу Arduino-платформи, яка буде використана (наприклад, Arduino Uno, Mega або Nano). Важливим аспектом є проектування схеми підключення сенсорів, модулів зв’язку та інших компонентів. Також необхідно продумати, як дані будуть передаватися для подальшої обробки, зберігання або візуалізації. Для цього можуть бути використані бездротові протоколи, такі як Wi-Fi або Bluetooth, а також технології, які забезпечують стабільність з'єднання.

3. Вибір компонентів: Після розробки архітектури системи здійснюється вибір конкретних компонентів. Цей етап включає не лише сенсори, але й модулі зв’язку, які забезпечують передачу даних. Наприклад, ESP8266 може бути вибрано для бездротового з’єднання з інтернетом, що дозволить здійснювати віддалений моніторинг. Вибір живлення також є важливим, оскільки для автономних систем

можуть використовуватися сонячні панелі або акумулятори, що забезпечать безперебійну роботу в умовах обмеженого доступу до електрики.

4. Складання прототипу: На цьому етапі відбувається складання прототипу системи. Процес включає фізичне підключення сенсорів до Arduino та налаштування схеми. Програмне забезпечення, написане для Arduino, повинно забезпечити зчитування даних з сенсорів, обробку отриманих даних та передачу їх на сервер або в систему для подальшої обробки. Важливо також інтегрувати систему візуалізації даних, що дозволить користувачам спостерігати за змінами в реальному часі.

5. Тестування системи: Після складання прототипу необхідно провести ретельне тестування системи. Це передбачає перевірку коректності роботи всіх сенсорів, точності отриманих даних, а також стабільності з'єднань. Тестування може включати проведення експериментів у різних умовах, щоб переконатися, що система справляється з різноманітними зовнішніми чинниками, такими як зміни температури, вологість, або навіть атмосферний тиск. Цей етап є критично важливим для виявлення і усунення можливих проблем, що можуть вплинути на точність даних.

6. Розгортання системи: Після успішного тестування прототипу здійснюється розгортання системи в обраній зоні моніторингу. Це може включати встановлення сенсорів на відповідних ділянках, модулів зв'язку та забезпечення доступу до джерела живлення. Розгортання повинно бути продумане, щоб уникнути впливу негативних умов, які можуть знизити ефективність роботи системи, таких як забруднення або фізичні перешкоди.

7. Моніторинг та обслуговування: Після розгортання системи необхідно забезпечити її моніторинг і обслуговування. Це передбачає регулярну перевірку працездатності сенсорів, оновлення програмного забезпечення, а також аналіз отриманих даних для оцінки ефективності системи. Моніторинг може здійснюватися через веб-інтерфейси або мобільні додатки, які дозволяють в реальному часі отримувати дані та реагувати на можливі аномалії. Технічне

обслуговування є важливим аспектом для забезпечення стабільності та безперебійної роботи системи.

8. Аналіз та вдосконалення системи: Останнім етапом є аналіз отриманих даних та вдосконалення системи на основі зворотного зв'язку. Це може включати доопрацювання алгоритмів обробки даних, покращення точності сенсорів або вдосконалення комунікаційних модулів для підвищення стабільності з’єднання. Регулярний аналіз результатів дозволяє виявляти нові тенденції, коригувати параметри роботи системи та адаптувати її до змінних умов навколишнього середовища[14].

Кожен з цих етапів є критично важливим для створення ефективної системи моніторингу навколишнього середовища на базі Arduino, яка не лише забезпечить збір та аналіз даних, а й стане основою для подальшого розвитку технологій екологічного моніторингу на підприємствах. Система повинна бути гнучкою і адаптивною, що дозволить їй реагувати на зміни у навколишньому середовищі та вимоги користувачів, тим самим підвищуючи її ефективність та корисність[15].

2.2. Вибір сенсорів для моніторингу навколишнього середовища

Для реалізації домашньої метеостанції, інтегрованої з системою обігріву, важливо вибрати відповідні сенсори, які дозволять ефективно моніторити навколишнє середовище. Основними сенсорами, що будуть використані у проекті, є:

1. Датчик температури, тиску та вологості BME280: BME280 — це високотехнологічний цифровий сенсор, що здатний вимірювати три важливі параметри: температуру, вологість і атмосферний тиск. Його компактний розмір і висока точність роблять його ідеальним вибором для проектів на основі Arduino, зокрема для домашніх метеостанцій. BME280 є вдосконаленою версією свого попередника BMP180 і забезпечує ще кращу точність та стабільність вимірювань.

Рисунок 2.1 - Датчик температури, тиску та вологості BME280

Характеристики:

- Температурний діапазон: від -40 до +85°C з точністю ±1°C.
- Вимірювання вологості: від 0 до 100% з точністю ±3% (в межах 20-80%).
- Атмосферний тиск: від 300 до 1100 hPa з точністю ±1 hPa.
- Інтерфейс: підтримує I2C та SPI.

Використання: Дані з BME280 використовуються для моніторингу погодних умов у реальному часі. Цей сенсор дозволяє отримувати актуальні показники температури, вологості та тиску, що є основою для адаптивного управління котлом. Наприклад, при зниженні температури зовні система автоматично активує обігрів. Інформація про вологість допомагає запобігти виникненню конденсату, що може бути шкідливим для приміщення та обладнання[16].

2. Модуль реального часу RTC DS3231

DS3231 — це високоточний модуль годинника реального часу, що забезпечує відстеження точного часу та дати. Він використовує спеціальний кварцовий резонатор, що дозволяє зберігати точність навіть при відключеній електроживленні.

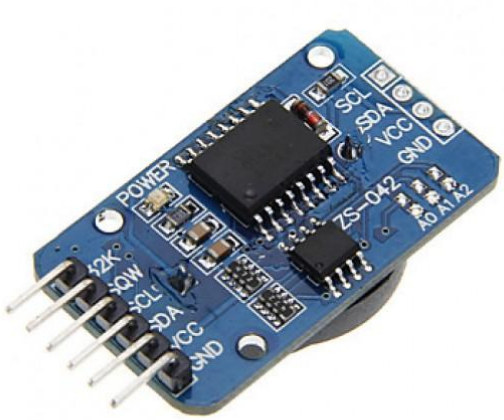

Рисунок 2.2 - Модуль реального часу RTC DS3231

Характеристики:

- Точність: ±2 хвилини на рік.
- Інтерфейс: підтримує I2C.
- Живлення: 3V - 5V.

Використання: RTC DS3231 використовується для фіксації часу і дати, що є важливим для ведення історії даних про погодні умови. Це дозволяє користувачу отримувати дані про зміни в навколишньому середовищі з точним часовим маркером. Наприклад, якщо система обігріву активується о 7:00 ранку, користувач може перевірити, яка була температура і вологість у цей час, що є корисним для аналізу ефективності обігріву.

3. Дисплей LCD2004 I2C

LCD2004 — це рідкокристалічний дисплей, що має 4 рядки по 20 символів. Він оснащений I2C інтерфейсом, що значно спрощує підключення та управління ним, зменшуючи кількість необхідних проводів.

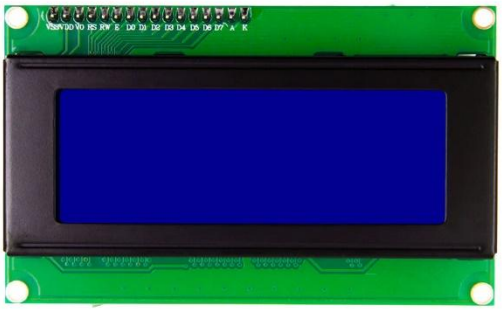

Рисунок 2.3 - Дисплей LCD2004 I2C

Характеристики:

- Розмір: 4x20 символів.
- Інтерфейс: I2C.
- Живлення: 5V.

Використання: LCD2004 використовується для відображення актуальних показників температури, вологості, атмосферного тиску та стану системи обігріву. Користувач може в реальному часі відстежувати дані, що дозволяє оперативно реагувати на зміни в навколишньому середовищі. Наприклад, якщо температура в приміщенні підвищується до небажаного рівня, на дисплеї може з’явитися попереджувальне повідомлення[17].

4. Модуль сенсорного ключа TTP223

TTP223 — це сенсорний модуль, який реагує на дотик, забезпечуючи інтуїтивне управління без необхідності фізичних кнопок. Його простота використання та компактний розмір роблять його ідеальним для інтеграції в домашні системи.

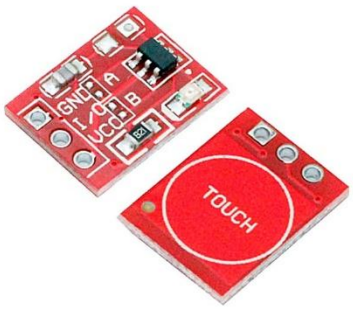

Рисунок 2.4 - Модуль сенсорного ключа TTP223

Характеристики:

- Робоча напруга: 2.0V - 5.5V.
- Споживана потужність: 0.5 мА в активному режимі.
- Розмір: компактний.

Використання: Модуль TTP223 використовується для управління системою обігріву через сенсорний дотик. Це дозволяє користувачу зручно включати чи вимикати обігрів, забезпечуючи зручність у використанні. Наприклад, якщо користувач хоче швидко підвищити температуру в кімнаті, він може просто доторкнутися до сенсора, що значно спрощує взаємодію з системою.

5. Підвищуючий перетворювач напруги DC-DC MT3608

Підвищуючий перетворювач напруги MT3608 використовується для збільшення напруги постійного струму в ланцюзі, а також для регулювання значення напруги. Цей модуль забезпечує стабільне підвищення напруги до 28 В і ідеально підходить для живлення пристроїв, які вимагають більшої напруги, ніж забезпечує джерело живлення[18].

Рисунок 2.5 - Підвищуючий перетворювач напруги DC-DC MT3608

Характеристики:

- Модель: HW-183 (MT3608)
- Вхідна напруга: 2 - 24 В постійного струму
- Вихідна напруга: 3 - 28 В постійного струму (регулювання за допомогою потенціометра)
- Максимальний вихідний струм: 2 А
- ККД: до 93%
- Робоча температура: -65 - 90 °C
- Розміри: 31 x 17 x 6 мм
- Вага: 4 г

Використання: Підвищуючий перетворювач MT3608 використовується у проекті для живлення компонентів системи моніторингу, які потребують вищої напруги. Завдяки можливості підключення до гнізда microUSB, перетворювач легко інтегрується в систему, спрощуючи процес підключення. Вхідна напруга може бути подана через термінали VIN + та VIN-, або через microUSB, що додає зручності у використанні.

2.3 Програмне забезпечення для системи моніторингу

Програмне забезпечення для комп’ютерної системи моніторингу навколишнього середовища на базі Arduino виконує ключову роль, адже саме воно дозволяє налагодити зв’язок між усіма компонентами, забезпечити збір та обробку даних, а також візуалізацію отриманих результатів. Arduino IDE, офіційне середовище розробки для платформи Arduino, є основним інструментом для програмування та налагодження роботи мікроконтролерів. Завдяки простому та інтуїтивно зрозумілому інтерфейсу, Arduino IDE підходить як для новачків, так і для професіоналів, дозволяючи з легкістю створювати, компілювати та завантажувати код безпосередньо на плату[19].

Arduino IDE використовує мову програмування, яка базується на C/C++. Це забезпечує високу гнучкість у розробці програмного забезпечення, дозволяючи використовувати широкий спектр конструкцій та алгоритмів для виконання різноманітних завдань. Важливо зазначити, що ця платформа також надає доступ до численних готових бібліотек, що дозволяє швидко підключати та використовувати різні типи сенсорів та модулів без необхідності писати код з нуля. Бібліотеки часто супроводжуються прикладами коду, що суттєво прискорює процес розробки[21].

Окрім Arduino IDE, існують і альтернативні платформи для розробки на базі Arduino, такі як PlatformIO та Visuino. PlatformIO інтегрується в потужний редактор Visual Studio Code та дозволяє зручніше керувати проєктом, підтримуючи не лише завантаження бібліотек, а й командну роботу над кодом завдяки системам контролю версій. Visuino, з іншого боку, є візуальним середовищем, яке особливо підходить для початківців. За допомогою графічного інтерфейсу в Visuino можна створювати програму, використовуючи блоки, кожен із яких відповідає за певну функцію — зчитування даних із сенсора, передача сигналу на реле, обробка інформації тощо. Такий підхід дозволяє уникнути написання коду вручну та зосередитися на логіці роботи системи[20].

Крім того, для зручності збору, зберігання та аналізу даних використовуються хмарні сервіси, такі як ThingSpeak та Blynk. ThingSpeak дозволяє збирати, аналізувати та візуалізувати дані з різних сенсорів у реальному часі, що забезпечує доступ до історії показників та спрощує їх аналіз. Blynk, у свою чергу, надає можливість створювати мобільні додатки для моніторингу даних, отриманих із системи, що дозволяє зручно контролювати стан навколишнього середовища зі смартфона.

Загалом, програмне забезпечення є одним із найважливіших елементів системи моніторингу, яке об’єднує всі компоненти в єдину функціональну систему, забезпечуючи обробку та передачу даних, а також зручність у їх використанні та аналізі[22].

Arduino IDE є потужним інструментом для розробки програмного забезпечення під платформу Arduino. Завдяки своїм можливостям, це середовище дозволяє створювати, редагувати, тестувати і вдосконалювати код для різних проєктів, включаючи комплексні системи моніторингу навколишнього середовища, автоматизацію процесів, інтеграцію з різноманітними сенсорами та інші IoT-додатки. Детальний опис основних можливостей Arduino IDE включає наступне:

1. **Редактор коду**: Arduino IDE забезпечує зручний текстовий редактор із підтримкою підсвітки синтаксису, що значно полегшує роботу над кодом. Підсвітка синтаксису допомагає виділяти різні частини коду — функції, змінні, команди — візуально, що робить код більш читабельним. Функція авто-доповнення не лише зменшує час на написання коду, але й запобігає багатьом синтаксичним помилкам. Це важливо при роботі з великими проектами, де складність коду може зрости, а авто-доповнення спрощує пошук потрібних функцій і змінних.

2. **Компіляція коду**: Компілятор, вбудований у Arduino IDE, відповідає за перетворення написаного коду у машинний, зрозумілий мікроконтролеру. На етапі компіляції IDE проводить перевірку коду на наявність синтаксичних помилок і логічних проблем, видаючи детальні повідомлення про помилки, які дозволяють користувачу їх швидко виправити. Процес компіляції забезпечує високу ефективність роботи і дозволяє впроваджувати складні обчислення та алгоритми в програмне забезпечення.

3. **Процес завантаження**: Arduino IDE дозволяє з легкістю завантажувати створений код на плату через USB-підключення. Інтуїтивний процес завантаження дозволяє тестувати код безпосередньо на платі, що робить можливим швидке відлагодження та перевірку функціональності. Це важливо для створення прототипів, де часті тести є ключем до успішного розвитку проєкту.

4. **Бібліотеки**: Arduino IDE надає доступ до величезного набору бібліотек, що розширюють функціональність системи та спрощують інтеграцію з різними апаратними компонентами. Користувачі можуть завантажувати бібліотеки для роботи з популярними сенсорами, дисплеями, бездротовими модулями зв'язку, що суттєво скорочує час на розробку. Наприклад, для сенсорів температури та вологості, таких як BME280, або модулів Wi-Fi, таких як ESP8266, існують готові бібліотеки, які полегшують налаштування та управління цими модулями.

5. **Дебагінг та серійний монітор**: Arduino IDE має серійний монітор, який використовується для виведення інформації в режимі реального часу. Це важливий інструмент для діагностики та налаштування програмного забезпечення, оскільки дозволяє отримувати дані безпосередньо з мікроконтролера і бачити результат виконання програми. Серійний монітор дозволяє відстежувати стан змінних, контролювати стан сенсорів та перевіряти коректність передавання даних, що значно полегшує процес усунення помилок.

6. **Приклади проєктів**: Arduino IDE надає набір вбудованих прикладів для швидкого ознайомлення з основами роботи на платформі Arduino. Ці

приклади охоплюють основні аспекти використання платформи, включаючи роботу з цифровими та аналоговими входами, керування світлодіодами, зчитування показників сенсорів та обробку даних. Вони особливо корисні для новачків, оскільки дозволяють легко зрозуміти принципи роботи мікроконтролера і надихнутися для створення власних проектів[23].

7. **Спільнота користувачів**: Arduino має одну з найбільших спільнот розробників серед платформ для мікроконтролерів. Спільнота постійно оновлює бібліотеки, ділиться кодами, надає підтримку в процесі розробки і ділиться інструкціями та навчальними матеріалами. Це дозволяє не тільки новачкам, але і професіоналам, вирішувати складні завдання, отримувати консультації та поради, а також ділитися власним досвідом. Спільнота також допомагає швидко орієнтуватися в оновленнях платформи та нових можливостях.

Таким чином, Arduino IDE забезпечує розробників усім необхідним для створення повноцінних проєктів — від написання та налагодження коду до інтеграції різних компонентів і діагностики системи в реальному часі.

Використання в проекті: У проекті домашньої метеостанції Arduino IDE використовується для написання коду, який забезпечує взаємодію з усіма компонентами системи. Це включає зчитування даних з датчика BME280, обробку даних від модуля RTC DS3231, відображення інформації на дисплеї LCD2004, а також управління сенсорним модулем TTP223. Код також включає логіку для комунікації з котлом, використовуючи підвищуючий перетворювач MT3608, що дозволяє налаштувати оптимальне живлення для системи[24].

Код, написаний в Arduino IDE, дозволяє здійснювати автоматичний моніторинг температури, вологості та тиску, що забезпечує користувачеві актуальну інформацію про погодні умови в приміщенні. Використовуючи отримані дані, система може регулювати обігрів, активуючи або деактивуючи котел в залежності від заданих параметрів.

Arduino IDE має низку переваг, що роблять його популярним вибором для розробників, зокрема для реалізації проектів, пов’язаних із домашніми автоматизаційними системами та системами моніторингу навколишнього середовища, такими як домашня метеостанція. Розглянемо детальніше ці переваги:

• Доступність: Arduino IDE є безкоштовним та сумісним з усіма основними операційними системами — Windows, macOS і Linux. Це значно розширює коло користувачів, дозволяючи кожному без фінансових витрат встановити та використовувати це середовище. Завдяки відсутності платного доступу, Arduino IDE є доступним для освітніх установ, студентів і хобістів, що сприяє поширенню технічних знань та популяризації електроніки.

• Гнучкість: Arduino IDE підтримує широкий спектр бібліотек і модулів, що робить його надзвичайно гнучким для різноманітних проектів. Це дозволяє розробникам легко адаптувати свої проекти під конкретні потреби, додаючи нові компоненти та пристрої. Наприклад, при створенні домашньої метеостанції можна з легкістю інтегрувати різні датчики для вимірювання температури, вологості, тиску та рівня ультрафіолетового випромінювання, що дозволяє системі збирати та передавати повну інформацію про стан довкілля.

• Документація: Arduino IDE має вичерпну документацію, що включає інструкції для налаштування середовища, написання коду та використання бібліотек. Документація також охоплює різні етапи розробки, від налаштування обладнання до виведення готового продукту, що робить її надзвичайно корисною для новачків. Завдяки цьому, користувачі можуть швидко освоїти принципи роботи з Arduino та перейти до розробки складніших проектів.

• Наявність ресурсів та активна спільнота: Arduino IDE має величезну спільноту користувачів та велику кількість ресурсів, що робить її ідеальною для початківців. Користувачі можуть знайти корисні інструкції, форуми, статті, відеоуроки та блоги, присвячені розробці проектів на основі Arduino. Це дозволяє розробникам дізнаватися про нові техніки та підходи, вирішувати проблеми, які

виникають під час роботи, і обмінюватися досвідом. Крім того, така спільнота сприяє розробці нових бібліотек та розширень, що розширюють можливості Arduino IDE та полегшують інтеграцію з різними модулями.

• Простота використання: Arduino IDE відома своєю простотою, яка робить її доступною навіть для тих, хто не має глибоких технічних знань. Інтерфейс IDE інтуїтивний і дозволяє швидко розпочати роботу без складних налаштувань. Це значно знижує бар'єр для входу, що важливо для освітніх проектів та домашнього використання, де потрібно швидко і легко розробити робочий прототип.

• Зручне тестування та налагодження: Завдяки серійному монітору, що вбудований у Arduino IDE, користувачі мають змогу тестувати код і відстежувати його роботу в режимі реального часу. Це дозволяє діагностувати помилки, переглядати стан змінних і виконання команд, що є критичним для проєктів, де важливо забезпечити стабільну та безпомилкову роботу системи.

• Широкий вибір прикладів: Arduino IDE постачається з великою кількістю вбудованих прикладів, які охоплюють базові та розширені функції платформи. Це особливо корисно для новачків, адже вони можуть швидко розпочати роботу, вивчаючи ці приклади, та використовувати їх як основу для своїх проектів.

У випадку реалізації проекту домашньої метеостанції, використання Arduino IDE надає низку переваг, які значно полегшують процес розробки. Середовище забезпечує зручне програмування, інтеграцію з сенсорами та налагодження, що робить його ідеальним інструментом для досягнення високої продуктивності та надійності системи[25].

Висновки до розділу 2

У другому розділі було детально розглянуто ключові етапи побудови системи моніторингу навколишнього середовища на базі Arduino, вибір необхідних сенсорів і відповідного програмного забезпечення.

На етапі побудови системи було визначено послідовність кроків, яка включає аналіз потреб системи, розробку архітектури, вибір обладнання, налаштування зв'язку між компонентами та їх інтеграцію для забезпечення повноцінного функціонування метеостанції. Обрані рішення дозволяють створити оптимальну структуру системи для ефективного моніторингу та зручної взаємодії з котлом.

Під час вибору сенсорів було проаналізовано різноманітні компоненти для вимірювання температури, вологості та тиску. Вибір сенсора BME280 було здійснено з урахуванням його надійності, багатофункціональності та здатності передавати точні дані, що є критичним для забезпечення якості моніторингу. Для підтримки точної часової синхронізації та коректного виведення даних на дисплей обрано модуль реального часу RTC DS3231 та дисплей LCD2004 I2C. Додатково було обрано сенсорний модуль TTP223 для інтуїтивного управління системою. Кожен з обраних компонентів був підібраний для досягнення максимальної ефективності та стабільності системи.

Для розробки програмного забезпечення було обрано Arduino IDE, що дозволяє інтегрувати всі компоненти в єдину систему. Це програмне забезпечення надає зручний інтерфейс, підтримує мову програмування C/C++, та забезпечує доступ до великої бібліотеки готових бібліотек для підключення сенсорів та модулів. Arduino IDE надає інструменти для тестування, компіляції та завантаження коду, що спрощує розробку і налаштування системи.

Таким чином, комплексний підхід до вибору інструментів та програмного забезпечення дозволяє створити ефективну та надійну систему моніторингу навколишнього середовища на базі Arduino, яка забезпечує точність даних та зручність у використанні.

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА НА БАЗІ ARDUINO

3.1. Архітектура системи моніторингу з функцією керування котлом

Архітектура системи моніторингу навколишнього середовища з функцією керування котлом на базі Arduino є складовою частиною сучасних автоматизованих систем управління домашнім мікрокліматом. У сучасному світі, де ефективність використання енергії та комфортність проживання мають величезне значення, розробка таких систем стає особливо актуальною. Основна мета цієї системи полягає в забезпеченні оптимальної температури в житлових та робочих приміщеннях, що дозволяє створити сприятливі умови для людей. Завдяки інтеграції датчиків температури, вологості та інших параметрів навколишнього середовища, система здійснює безперервний моніторинг і аналіз цих даних[26].

Ця система не лише контролює температуру, а й оптимізує витрати енергії, що є критично важливим у контексті зростаючих тарифів на електрику та газ. Використовуючи алгоритми керування на основі даних з сенсорів, система може адаптувати роботу котла в залежності від потреб користувачів і змін в навколишньому середовищі. Наприклад, якщо температура в приміщенні падає нижче встановленого порогу, система автоматично активує котел, забезпечуючи швидке нагрівання. У випадку, коли температура перевищує задані межі, котел буде вимкнений, що запобігає його перевантаженню і знижує ризик поломок.

Крім того, реалізація гістерезису в управлінні котлом дозволяє уникнути частих циклів вмикання і вимикання, що є особливо важливим для збереження довговічності обладнання. Постійне перемикання може призвести до швидшого зносу компонентів, тому система розроблена таким чином, щоб забезпечити оптимальну стабільність температури з урахуванням заданих параметрів. В результаті, така архітектура системи не лише підвищує комфортність в приміщеннях, але й економить ресурси, що робить її вигідною для користувачів.

Завдяки інтеграції новітніх технологій, таких як бездротові комунікаційні модулі, система стає ще більш зручною у використанні, дозволяючи користувачам контролювати температурний режим дистанційно через смартфони або комп'ютери. Таким чином, архітектура системи моніторингу навколишнього середовища з функцією керування котлом на базі Arduino є ефективним інструментом для досягнення комфортного мікроклімату в приміщеннях, оптимізації витрат енергії та підвищення надійності обігрівальних систем[27].

Основні компоненти системи

Система моніторингу навколишнього середовища з функцією керування котлом на базі Arduino складається з кількох ключових компонентів, кожен з яких виконує специфічну роль у забезпеченні стабільної та ефективної роботи всієї системи.

1. **Датчики**: Основним елементом системи є датчики, які забезпечують збирання інформації про параметри навколишнього середовища. В нашій системі використовується сенсор BME280, здатний вимірювати температуру, вологість і атмосферний тиск. Цей датчик працює на основі цифрового інтерфейсу I2C, що дозволяє легко інтегрувати його з Arduino. Регулярне зчитування даних із сенсора дозволяє системі оперативно реагувати на зміни в навколишньому середовищі. Важливою особливістю є те, що BME280 має високу точність та стабільність, що робить його ідеальним вибором для домашніх метеостанцій.

2. **Arduino метеостанція**: Перше Arduino, яке використовується в системі, виконує функцію метеостанції. Воно відповідає за зчитування даних із датчиків, обробку отриманих значень і передачу інформації на сервер. Це Arduino може бути оснащене додатковими модулями для підключення до Wi-Fi, такими як ESP8266 або ESP32, що дозволяє системі зберігати зв'язок із сервером, навіть якщо метеостанція знаходиться в різних частинах будинку. Завдяки такій архітектурі, користувач може отримувати актуальні дані про стан навколишнього середовища в режимі реального часу.

3. **Сервер**: Сервер на основі другого Arduino виконує важливу роль у системі, отримуючи дані з метеостанції і приймаючи рішення на основі отриманих показників. Сервер обробляє інформацію, порівнюючи температуру з встановленими пороговими значеннями, і приймає рішення про активацію або деактивацію котла. Для зв'язку з метеостанцією сервер використовує прості HTTP-запити, що робить комунікацію між компонентами легкою та зрозумілою. У разі перевищення або недостачі температури, сервер може автоматично відправляти команди на реле для управління котлом.

4. **Реле**: Реле в системі є елементом, що виконує роль електричного перемикача, дозволяючи Arduino контролювати живлення котла. Воно підключається до вхідних контактів котла та, отримуючи сигнали від сервера, відкриває або закриває електричний ланцюг. Завдяки реле, система може вмикати або вимикати котел у відповідь на зміни температури, забезпечуючи автоматизовану функцію управління. Використання реле дозволяє безпосередньо взаємодіяти з електричними компонентами системи, що робить управління котлом безпечним і надійним.

5. **Котел**: Котел є основним джерелом обігріву в системі, і він повинен бути сумісним з реле для забезпечення правильної роботи. У системі котел підключається до реле, що контролює його живлення. Завдяки інтеграції котла з системою моніторингу, користувач отримує можливість дистанційно контролювати температуру в квартирі, а також зменшити витрати на енергію. Котел автоматично вмикається або вимикається залежно від даних, отриманих від метеостанції, що підвищує загальну ефективність системи.

Взаємодія між компонентами

Уся система базується на бездротовій технології, що забезпечує гнучкість у розташуванні компонентів. Взаємодія між Arduino, датчиками і реле відбувається через цифрові та аналогові виходи. Сервер обробляє дані за допомогою простих HTTP-запитів, що робить систему легкою у налаштуванні і масштабуванні[28].

Ця архітектура дозволяє створити ефективну і надійну систему моніторингу навколишнього середовища, яка не тільки забезпечує контроль за параметрами середовища, але й автоматизує управління обігрівом, що позитивно впливає на енергоефективність і комфортність проживання.

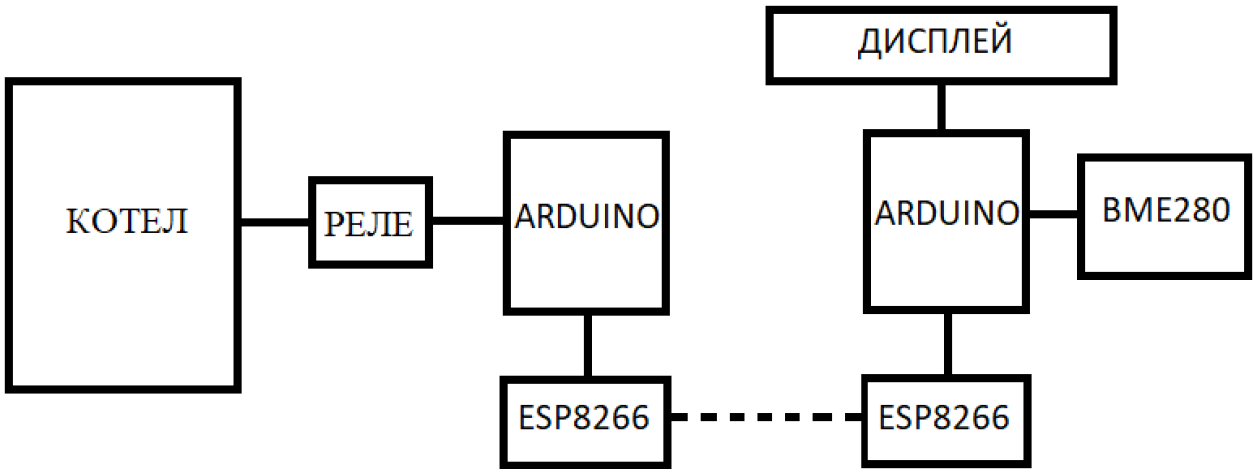

Рисунок 3.1 - Схема системи моніторингу та управління котлом на базі Arduino

3.2 Вибір та налаштування компонентів для управління котлом

3.2.1. Wi-Fi модуль ESP8266

Вибір і обґрунтування: Wi-Fi модуль ESP8266 обрано для забезпечення бездротового зв'язку в системі моніторингу та управління котлом. Цей модуль є популярним через свою доступність, простоту використання та можливість підключення до Wi-Fi мережі. ESP8266 має вбудований мікроконтролер, що дозволяє виконувати обробку даних безпосередньо на модулі, зменшуючи навантаження на основний контролер Arduino.

- **Переваги ESP8266:**

  Компактний розмір і легкість в інтеграції.

  Наявність бібліотек для Arduino, що спрощує програмування.

Можливість роботи з веб-сервісами для віддаленого моніторингу.

Опис налаштувань для зв'язку з котлом:

1. **Підключення:** ESP8266 підключається до Arduino через UART інтерфейс (RX до TX Arduino та TX до RX Arduino).

2. **Налаштування Wi-Fi:**

- У програмі для Arduino необхідно імплементувати бібліотеку ESP8266WiFi для роботи з модулем.

- Вказати SSID та пароль вашої Wi-Fi мережі, використовуючи наступний код:

```
#include <ESP8266WiFi.h>

const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

void setup() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}
```

3. **Обмін даними:**

- Для управління котлом через модуль ESP8266 можна використовувати HTTP запити або WebSocket для передачі даних про стан котла (включено/вимкнено).

- Реалізація віддаленого доступу до системи через веб-сервер, налаштований на ESP8266, дозволяє користувачеві моніторити і контролювати котел з мобільних пристроїв.

3.2.2. Реле для керування котлом

Реле є одним із ключових компонентів у системі автоматизації котла, оскільки воно відповідає за управління електроживленням. Воно діє як електричний перемикач, що дозволяє контролювати потік електрики до котла на основі сигналів, отриманих від контролера Arduino. Таким чином, реле забезпечує можливість автоматичного вмикання та вимикання котла відповідно до температурних показників, що допомагає підтримувати оптимальний мікроклімат у приміщенні[29].

Принцип роботи

1. **Збір даних від датчика:** У системі використовується температурний сенсор (наприклад, BME280), який постійно вимірює температуру в приміщенні. Датчик підключається до Arduino, яка обробляє отримані дані.

2. **Прийняття рішення:** Програма на Arduino постійно моніторить значення температури. Якщо виміряна температура нижча за задану порогову величину (desiredTemperature), Arduino приймає рішення про активацію реле.

3. **Активація реле:**

   Вмикання котла: Arduino подає високий сигнал (HIGH) на реле, що призводить до замикання контактів реле. Це дозволяє електричному струму проходити до котла, вмикаючи його.

   Вимикання котла: Коли температура досягає заданого рівня, Arduino подає низький сигнал (LOW) на реле, розмикаючи контакти. Це відключає котел, запобігаючи його перегріву та економлячи електроенергію.

Налаштування реле

1. **Підключення реле:**

Вибір реле: Рекомендується використовувати реле, розраховане на струм не менше 10А. Це дозволить безпечно керувати потужними котлами, які можуть споживати значну кількість електрики.

Схема підключення: Реле підключається до одного з цифрових виходів Arduino (наприклад, до піну 7). Також реле повинно мати підключення до джерела живлення (зазвичай 5В) та загальний контакт, що з'єднує ланцюг[30].

Безпека: Використання реле з відповідною ізоляцією важливе для захисту компонентів системи від перевантаження і короткого замикання.

2. **Програмування реле:**

У коді для Arduino необхідно налаштувати пін для управління реле та імплементувати логіку, яка активує або деактивує реле на основі значення температури.

```
const int relayPin = 7; // Вказуємо пін для підключення реле
const int temperatureSensorPin = A0; // Вказуємо пін для підключення сенсора
float desiredTemperature = 22.0; // Задане значення температури

void setup() {
  pinMode(relayPin, OUTPUT); // Налаштування піну реле як виходу
  Serial.begin(9600); // Ініціалізація серійного з'єднання
}

void loop() {
  float temperature = readTemperature(); // Функція для зчитування температури

  if (temperature < desiredTemperature) {
    digitalWrite(relayPin, HIGH); // Включити котел
  } else {
    digitalWrite(relayPin, LOW); // Вимкнути котел
  }
```

```
  delay(1000); // Затримка для уникнення перевантаження процесора
}

float readTemperature() {
  // Функція для зчитування значення температури з сенсора
  // (залежно від використовуваного датчика)
  return analogRead(temperatureSensorPin) * (5.0 / 1023.0); // Простий приклад
}
```

3.3 Програмне забезпечення системи навколишнього середовища

Для отримання температури з домашньої метеостанції на одному Arduino та передачі її на інше Arduino, яке керуватиме котлом, використовується модуль ESP8266 для бездротової передачі даних.

3.3.1 Програмування Arduino для збору інформації

Перше Arduino зчитує температуру з датчика BME280 і передає її по Wi-Fi через модуль ESP8266.

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// Налаштування Wi-Fi
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
const char* serverUrl = "http://192.168.1.100/temperature"; // IP-адреса другого Arduino

Adafruit_BME280 bme;

void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Підключення до WiFi...");
  }
  Serial.println("Підключено до WiFi");

  if (!bme.begin(0x76)) {
    Serial.println("Не знайдено BME280");
    while (1);
  }
}

void loop() {
  float temperature = bme.readTemperature();
  sendTemperature(temperature);
  delay(5000); // Відправка кожні 5 секунд
}

void sendTemperature(float temperature) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(serverUrl);

    String postData = "temperature=" + String(temperature);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    int httpResponseCode = http.POST(postData);

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Відправлено температуру: " + String(temperature));
      Serial.println("Відповідь сервера: " + response);
    } else {
      Serial.println("Помилка при відправці даних");
    }
    http.end();
  }
}
```

3.3.2 Програмування Arduino для керування котлом

Друге Arduino, також з ESP8266 або ESP32, отримує температуру через Wi-Fi та використовує логіку гістерезису для керування реле.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

// Налаштування Wi-Fi
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

ESP8266WebServer server(80);

const int relayPin = 7; // Пін для реле
float currentTemperature = 0;
float lowerThreshold = 21.0;
float upperThreshold = 23.0;
bool isHeaterOn = false;

void setup() {
  Serial.begin(9600);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH); // Ініціалізація реле у вимкненому стані

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Підключення до WiFi...");
  }
  Serial.println("Підключено до WiFi");

  server.on("/temperature", HTTP_POST, handleTemperatureUpdate);
  server.begin();
  Serial.println("Сервер запущено");
}

void loop() {
  server.handleClient();
```

```
  // Логіка гістерезису
  if (currentTemperature < lowerThreshold && !isHeaterOn) {
    digitalWrite(relayPin, LOW); // Увімкнути реле
    isHeaterOn = true;
    Serial.println("Котел увімкнено");
  } else if (currentTemperature > upperThreshold && isHeaterOn) {
    digitalWrite(relayPin, HIGH); // Вимкнути реле
    isHeaterOn = false;
    Serial.println("Котел вимкнено");
  }

  delay(1000);
}

void handleTemperatureUpdate() {
  if (server.hasArg("temperature")) {
    currentTemperature = server.arg("temperature").toFloat();
    Serial.println("Отримано нову температуру: " + String(currentTemperature));
  }
  server.send(200, "text/plain", "Температура отримана");
}
```

Перше Arduino (метеостанція) зчитує температуру з BME280 і надсилає її через POST-запит на IP-адресу другого Arduino. Друге Arduino (керування котлом) працює як сервер, приймаючи значення температури та використовуючи логіку гістерезису для керування реле.

Wi-Fi з'єднання між обома Arduino забезпечує бездротову передачу даних, використовуючи ESP8266. Цей підхід дозволяє автоматично керувати котлом на основі температури з метеостанції.

Висновки до розділу 3

У цьому розділі представлено всебічний огляд розробки системи моніторингу навколишнього середовища на базі Arduino з функцією управління котлом. Розроблена система є інтегрованим рішенням, яке об’єднує кілька ключових компонентів для забезпечення ефективного збору, обробки та передачі даних, а також для автоматизації процесів управління котлом, що безпосередньо впливає на комфорт у житлових приміщеннях.

Архітектура системи складається з декількох елементів, зокрема сенсорів, контролерів та модулів зв'язку. Wi-Fi модуль ESP8266 виконує важливу роль у бездротовій передачі даних між домашньою метеостанцією і пристроєм, що контролює котел, що значно спрощує інтеграцію різних компонентів та забезпечує гнучкість у налаштуванні системи. Завдяки цьому досягається ефективне управління енергоспоживанням і зменшуються витрати на обігрів.

Реле, що використовується для вмикання та вимикання котла, є ще одним важливим елементом системи. Завдяки своїй конструкції реле забезпечує стабільну роботу котла, реагуючи на отримані дані з температурного датчика. Встановлення логіки гістерезису у програмному забезпеченні дозволяє уникнути частих вмикань та вимикань, що може негативно вплинути на тривалість роботи котла, та забезпечує більш плавне підтримання комфортного рівня температури в приміщенні.

Налаштування програмного забезпечення для обох Arduino створює чіткий алгоритм, який обробляє показники температури та здійснює управління котлом. Це дозволяє автоматизувати весь процес контролю, зменшуючи потребу в ручному втручанні та забезпечуючи більш точне реагування на зміни температури.

Загалом, розроблена система моніторингу навколишнього середовища на базі Arduino не лише підвищує комфорт у житлових приміщеннях, але також сприяє зниженню витрат на енергію, що особливо актуально в умовах зростаючих цін на енергоносії. У майбутньому є можливості для розширення функціоналу системи шляхом інтеграції додаткових сенсорів, таких як датчики вологості або якості

повітря, що дозволить створити більш комплексну та адаптивну систему управління.

Упровадження цієї технології в домашніх умовах може стати важливим кроком у напрямку енергоефективності та екологічної свідомості, адже автоматизація системи обігріву сприяє не лише економії ресурсів, але й поліпшенню загального рівня комфорту та безпеки. Розробка даної системи відкриває нові горизонти для досліджень та інновацій у галузі автоматизації домашнього господарства, що може мати позитивний вплив на якість життя користувачів.

ЗАГАЛЬНІ ВИСНОВКИ

Кваліфікаційна робота присвячена розробці системи моніторингу навколишнього середовища на базі Arduino, що інтегрує функцію управління котлом. У результаті дослідження було визначено, що впровадження таких систем має значний потенціал у підвищенні якості життя користувачів та зниженні витрат на енергію.

У першому розділі було проведено всебічний аналіз комп'ютерних систем моніторингу навколишнього середовища. Основну увагу було приділено структурі, складовим елементам та функціональним можливостям існуючих рішень. Було виявлено, що системи на базі Arduino відзначаються не лише простотою реалізації, але й високою доступністю компонентів, що сприяє їх широкому використанню в різних сферах, таких як сільське господарство, екологічний моніторинг та промисловість. Ці системи забезпечують можливість своєчасного збору та обробки даних про стан навколишнього середовища, що дозволяє оперативно реагувати на зміни та ухвалювати зважені рішення.

Другий розділ роботи детально висвітлює етапи побудови системи моніторингу, починаючи від аналізу потреб та розробки архітектури до вибору сенсорів і налаштування зв'язку між компонентами. Ретельний вибір сенсорів, таких як BME280, демонструє важливість використання надійних та точних пристроїв для вимірювання температури, вологості та тиску. Інші елементи, такі як модуль реального часу RTC DS3231 і дисплей LCD2004 I2C, були вибрані для забезпечення коректного виведення даних і підтримки точної часової синхронізації. Застосування сенсорного модуля TTP223 для управління системою підвищує зручність взаємодії користувача з пристроєм. Вибір Arduino IDE як основного програмного забезпечення забезпечив інтеграцію всіх компонентів у єдину систему, спростивши процес розробки та тестування.

Третій розділ розкриває результати реалізації розробленої системи моніторингу, підкреслюючи її ефективність у покращенні комфорту в житлових приміщеннях та зниженні енергетичних витрат. Важливу роль у цьому процесі

відіграє Wi-Fi модуль ESP8266, який забезпечує бездротову передачу даних, дозволяючи інтегрувати різні компоненти системи і надаючи можливість віддаленого доступу. Використання реле для управління котлом гарантує стабільну роботу системи, адже реле реагує на дані з температурного датчика, забезпечуючи автоматизовану підтримку комфортного рівня температури в приміщенні. Впровадження логіки гістерезису у програмному забезпеченні зменшує частоту вмикань та вимикань котла, що позитивно впливає на його експлуатаційні характеристики.

Загалом, результати роботи підтверджують актуальність та доцільність використання комп'ютерних систем моніторингу навколишнього середовища на базі Arduino. Розроблена система не лише підвищує комфорт у житлових приміщеннях, але й активно сприяє енергоефективності, що особливо важливо в умовах зростаючих цін на енергоресурси. Впровадження автоматизованих систем обігріву дозволяє досягти значної економії ресурсів та покращення загального рівня безпеки та комфорту.

Подальші перспективи розвитку цієї технології можуть включати інтеграцію додаткових сенсорів, таких як датчики вологості або якості повітря, що дозволить створити більш комплексні та адаптивні системи управління. Розробка таких систем може відкрити нові горизонти для досліджень і інновацій у галузі автоматизації домашнього господарства, що потенційно може призвести до суттєвих покращень у якості життя користувачів.

Впровадження розробленої технології у повсякденне життя користувачів не лише позитивно вплине на ефективність використання енергетичних ресурсів, але й сприятиме підвищенню екологічної свідомості. Зважаючи на глобальні виклики, пов'язані з енергетичною безпекою та змінами клімату, автоматизація системи обігріву може стати важливим кроком у напрямку сталого розвитку та екологічної відповідальності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Getting Started with Arduino: The Open Source Electronics Prototyping Platform by Massimo Banzi and Michael Shiloh. O'Reilly Media, 2014. 240 p.
2. Arduino Cookbook by Michael Margolis and Brian Evans. O'Reilly Media, 2011. 400 p.
3. Arduino Project Handbook: 25 Practical Projects to Get You Started by Mark Geddes. Rocky Nook, 2018. 236 p.
4. Programming Arduino: Getting Started with Sketches by Simon Monk. McGraw-Hill Education, 2016. 272 p.
5. Building Arduino Projects for the Internet of Things: Connect Your Projects to the Cloud by Jeremy Blum and Daniel J. Watts. Wiley, 2016. 288 p.
6. Arduino for Dummies by John Nussey. Wiley, 2015. 360 p.
7. Hands-On Internet of Things with Arduino: Build Exciting IoT Projects with Arduino and ESP8266 by Dmitry M. Vinnik. Packt Publishing, 2018. 274 p.
8. Make: Arduino Bots and Gadgets by Instructables.com. Maker Media, 2013. 274 p.
9. Sensor Technologies for Civil Infrastructures by Jiangyun Feng. Springer, 2019. 175 p.
10. Environmental Monitoring with Arduino: A Practical Guide to Building a Wireless Sensor Network by David J. Johnson. Apress, 2017. 197 p.
11. Internet of Things with Arduino Blueprints by Harprit Singh. Packt Publishing, 2014. 374 p.
12. Arduino Robotics by John-David Warren, Josh Adams, and Harald M. Schmitt. Packt Publishing, 2015. 392 p.
13. Practical Electronics for Inventors by Paul Scherz and Simon Monk. McGraw-Hill Education, 2016. 672 p.
14. Data Acquisition and Control with Arduino by Robert J. Smith. Springer, 2018. 190 p.

15. Mastering Arduino by James M. McRoberts. Packt Publishing, 2017. 248 p.
16. Arduino and Raspberry Pi Sensor Projects for the Evil Genius by Mike Chelen. McGraw-Hill Education, 2016. 224 p.
17. Beginning C for Arduino by Brian Evans. Apress, 2016. 240 p.
18. Exploring Arduino: Tools and Techniques for Engineering Wizardry by Jeremy Blum. Wiley, 2019. 384 p.
19. Arduino Programming in 24 Hours, Sams Teach Yourself by Richard Blum and Christine Bresnahan. Sams Publishing, 2014. 432 p.
20. Arduino Workshop: A Hands-On Introduction with 65 Projects by John Boxall. No Starch Press, 2019. 392 p.
21. The Maker's Guide to the Zombie Apocalypse: Defend Your Base with Simple Circuits, Arduino, and Raspberry Pi by Simon Monk. No Starch Press, 2015. 280 p.
22. Environmental Monitoring with Arduino: Building Simple Devices to Collect Data About the World Around Us by Emily Gertz and Patrick Di Justo. Maker Media, 2012. 100 p.
23. Arduino Wearable Projects by Tony Olsson. Packt Publishing, 2015. 200 p.
24. Beginning Sensor Networks with Arduino and Raspberry Pi by Charles Bell. Apress, 2016. 403 p.
25. Arduino Internals by Dale Wheat. Apress, 2011. 272 p.
26. Programming the Internet of Things with Node.js and Raspberry Pi by Rajesh Singh, Anita Gehlot, and Lovi Raj Gupta. CRC Press, 2018. 224 p.
27. Internet of Things Programming with JavaScript by Ruben Oliva Ramos. Packt Publishing, 2017. 300 p.
28. Arduino and Kinect Projects: Design, Build, Blow Their Minds by Enrique Ramos Melgar and Ciriaco Castro Diez. Apress, 2012. 372 p.
29. IoT Projects with Arduino Nano 33 BLE Sense by Agus Kurniawan. Packt Publishing, 2020. 220 p.
30. Learning Internet of Things by Peter Waher. Packt Publishing, 2015. 340 p.
31. Sensors for Mechatronics by Paul P.L. Regtien. Elsevier, 2018. 300 p.

32. Environmental Data Analysis and Modeling by John Wainwright and Mark Mulligan. Wiley-Blackwell, 2013. 496 p.
33. Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things by Fotios Chantzis et al. No Starch Press, 2020. 304 p.
34. The Internet of Things: Key Applications and Protocols by Olivier Hersent, David Boswarthick, and Omar Elloumi. Wiley, 2012. 360 p.
35. IoT Solutions in Microsoft's Azure IoT Suite: Data Acquisition and Analysis in the Real World by Scott Klein and Sean McGehee. Apress, 2017. 250 p.

ДОДАТОК А

Основний код програми на мові C++ для мікроконтролера ATMEGA328p:

```
void checkBrightness() {
  if (LCD_BRIGHT == 11) {
    if (analogRead(PHOTO) < BRIGHT_THRESHOLD) {
      analogWrite(BACKLIGHT, LCD_BRIGHT_MIN);
    } else {
      analogWrite(BACKLIGHT, LCD_BRIGHT_MAX);
    }
  } else {
    analogWrite(BACKLIGHT, LCD_BRIGHT * LCD_BRIGHT * 2.5);
  }

  if (LED_BRIGHT == 11) {
    if (analogRead(PHOTO) < BRIGHT_THRESHOLD) {
#if (LED_MODE == 0)
      LED_ON = (LED_BRIGHT_MIN);
#else
      LED_ON = (255 - LED_BRIGHT_MIN);
#endif
    } else {
#if (LED_MODE == 0)
      LED_ON = (LED_BRIGHT_MAX);
#else
      LED_ON = (255 - LED_BRIGHT_MAX);
#endif
    }
  }
}



void modesTick() {
  button.tick();
  boolean changeFlag = false;
  if (button.isSingle()) {
    if (mode >= 240) {
      podMode++;
      switch (mode) {
        case 252:
          //       podMode++;
          if (podMode > 4) podMode = 0;
          LEDType = podMode;
          changeFlag = true;
          break;

        case 253:
          //       podMode++;
          if (podMode > 11) podMode = 0;
          LCD_BRIGHT = podMode;
          checkBrightness();
          changeFlag = true;
          break;

        case 254:
          //       podMode++;
          if (podMode > 11) podMode = 0;
          LED_BRIGHT = podMode;
          changeFlag = true;
          break;

        case 255:
          //       podMode++;
          if (podMode > 15) podMode = 1;
          changeFlag = true;
          break;
      }
    } else {
      do {
        mode++;
        if (mode > 10) mode = 0;
```

```
#if (CO2_SENSOR == 0 && mode == 1)
      mode = 3;
#endif
    } while (((VIS_ONDATA & (1 << (mode - 1))) == 0) && (mode > 0));
    changeFlag = true;
  }
 }
 if (button.isDouble()) {
  if (mode > 0 && mode < 11) {
   MAX_ONDATA = (int)MAX_ONDATA ^ (1 << (mode - 1));
  } else if (mode == 0)  {
   mode0scr++;
   if (CO2_SENSOR == 0 && mode0scr == 1) mode0scr++;
   if (mode0scr > 5) mode0scr = 0;
  } else if (mode > 240) podMode = 1;
  changeFlag = true;
 }

 if ((button.isTriple()) && (mode == 0)) {
  mode = 255;
  podMode = 3;
  changeFlag = true;
 }

 if (button.isHolded()) {

  switch (mode) {
   case 0:
    bigDig = !bigDig;
    break;
   case 252:
    mode = 255;
    podMode = 1;
    break;
   case 253:
    mode = 255;
    podMode = 1;
    break;
   case 254:
    mode = 255;
    podMode = 1;
    break;
   case 255:
    if (podMode == 2 || podMode == 1) mode = 0;
    if (podMode >= 3 && podMode <= 5) mode = 255 - podMode + 2;
    if (podMode >= 6 && podMode <= 17) VIS_ONDATA = VIS_ONDATA ^ (1 << (podMode - 6));
    if (podMode == 1) {
     if (EEPROM.read(2) != (MAX_ONDATA & 255)) EEPROM.write(2, (MAX_ONDATA & 255));
     if (EEPROM.read(3) != (MAX_ONDATA >> 8)) EEPROM.write(3, (MAX_ONDATA >> 8));
     if (EEPROM.read(4) != (VIS_ONDATA & 255)) EEPROM.write(4, (VIS_ONDATA & 255));
     if (EEPROM.read(5) != (VIS_ONDATA >> 8)) EEPROM.write(5, (VIS_ONDATA >> 8));
     if (EEPROM.read(6) != mode0scr) EEPROM.write(6, mode0scr);
     if (EEPROM.read(7) != bigDig) EEPROM.write(7, bigDig);
     if (EEPROM.read(8) != LED_BRIGHT) EEPROM.write(8, LED_BRIGHT);
     if (EEPROM.read(9) != LCD_BRIGHT) EEPROM.write(9, LCD_BRIGHT);
     if (EEPROM.read(10) != LEDType) EEPROM.write(10, LEDType);
     if (EEPROM.read(0) != 122) EEPROM.write(0, 122);
    }
    if (podMode < 6) podMode = 1;
    if (mode == 252) podMode = LEDType;
    if (mode == 254) podMode = LED_BRIGHT;
    if (mode == 253) podMode = LCD_BRIGHT;
    break;
   default:
    mode = 0;
  }
  changeFlag = true;
 }

 if (changeFlag) {
```

```
  if (mode >= 240) {
    lcd.clear();
    lcd.createChar(1, BM);  //ь
    lcd.createChar(2, IY);  //Й
    lcd.createChar(3, DD);  //Д
    lcd.createChar(4, II);  //И
    lcd.createChar(5, IA);  //Я
    lcd.createChar(6, YY);  //Ы
    lcd.createChar(7, AA);  //Э
    lcd.createChar(0, ZZ);  //Ж
    lcd.setCursor(0, 0);
  }
  if (mode == 255) {
#if (WEEK_LANG == 1)
    lcd.print("HACTPO\2K\4:");
#else
    lcd.print("Setup:");
#endif
    lcd.setCursor(0, 1);
    switch (podMode) {
      case 1:
#if (WEEK_LANG == 1)
        lcd.print("COXPAH\4T\1");
#else
        lcd.print("Save");
#endif
        break;
      case 2:
#if (WEEK_LANG == 1)
        lcd.print("B\6XO\3");
#else
        lcd.print("Exit");
#endif
        break;
      case 5:
#if (WEEK_LANG == 1)
        lcd.print("PE\10.\4H\3\4KATOPA");
#else
        lcd.print("indicator mode");
#endif
        break;
      case 3:
#if (WEEK_LANG == 1)
        lcd.print("\5PK.\4H\3\4KATOPA");
#else
        lcd.print("indicator brt.");
#endif
        break;
      case 4:
#if (WEEK_LANG == 1)
        lcd.print("\5PK.\7KPAHA");
#else
        lcd.print("Bright LCD");
#endif
        break;
    }
    if (podMode >= 6 && podMode <= 17) {
      lcd.createChar(8, FF);  //ф
      lcd.createChar(7, GG);  //Г
      lcd.createChar(5, LL);  //Л
      lcd.setCursor(10, 0);
#if (WEEK_LANG == 1)
      lcd.print("\7PA\10\4KOB");
#else
      lcd.print("Charts  ");
#endif
      lcd.setCursor(0, 1);
      if ((3 & (1 << (podMode - 6))) != 0) lcd.print("CO2 ");
      if ((12 & (1 << (podMode - 6))) != 0) {
#if (WEEK_LANG == 1)
        lcd.print("B\5,% ");
#else
        lcd.print("Hum,%");
#endif
      }
      if ((48 & (1 << (podMode - 6))) != 0) lcd.print("t\337 ");
```

```
      if ((192 & (1 << (podMode - 6))) != 0) {
       if (PRESSURE) lcd.print("p,rain ");
       else lcd.print("p,mmPT ");
      }
      if ((768 & (1 << (podMode - 6))) != 0) {
#if (WEEK_LANG == 1)
       lcd.print("B\6C,m  ");
#else
       lcd.print("hgt,m  ");
#endif
      }

      if ((1365 & (1 << (podMode - 6))) != 0) {
       lcd.createChar(3, CH);  //Ч
       lcd.setCursor(8, 1);
#if (WEEK_LANG == 1)
       lcd.print("\3AC:");
#else
       lcd.print("Hour:");
#endif
      } else {
       lcd.setCursor(7, 1);
#if (WEEK_LANG == 1)
       lcd.print("\3EH\1:");
#else
       lcd.print("Day: ");
#endif
      }
      if ((VIS_ONDATA & (1 << (podMode - 6))) != 0) {
#if (WEEK_LANG == 1)
       lcd.print("BK\5 ");
#else
       lcd.print("On  ");
#endif
      }
      else {
#if (WEEK_LANG == 1)
       lcd.print("B\6K\5");
#else
       lcd.print("Off ");
#endif
      }
     }
    }
    if (mode == 252) {
     LEDType = podMode;
     lcd.createChar(6, LL);  //Л
     lcd.createChar(3, DD);  //Д
     lcd.createChar(5, II);  //И
     lcd.createChar(8, ZZ);  //Ж
     lcd.setCursor(0, 0);
#if (WEEK_LANG == 1)
     lcd.print("PE\10.\4H\3\4KATOPA:");
#else
     lcd.print("indicator mode:");
#endif
     lcd.setCursor(0, 1);
     switch (podMode) {
      case 0:
       lcd.print("CO2   ");
       break;
      case 1:
#if (WEEK_LANG == 1)
       lcd.print("B\6A\10H.");
#else
       lcd.print("Humid.");
#endif
       break;
      case 2:
       lcd.print("t\337    ");
       break;
      case 3:
#if (WEEK_LANG == 1)
       lcd.print("OCA\3K\5");
#else
       lcd.print("rain  ");
```

```
#endif
        break;
      case 4:
#if (WEEK_LANG == 1)
        lcd.print("\3AB\6EH\5E");
#else
        lcd.print("pressure");
#endif
        break;
    }

  }
  if (mode == 253) {
#if (WEEK_LANG == 1)
    lcd.print("\5PK.\7KPAHA:");// + String(LCD_BRIGHT * 10) + "%
");
#else
    lcd.print("Bright LCD:");
#endif
    //lcd.setCursor(11, 0);
    if (LCD_BRIGHT == 11) {
#if (WEEK_LANG == 1)
      lcd.print("ABTO ");
#else
      lcd.print("Auto ");
#endif
    }
    else lcd.print(String(LCD_BRIGHT * 10) + "%");
  }
  if (mode == 254) {
#if (WEEK_LANG == 1)
    lcd.print("\5PK.\4H\3\4K.:");// + String(LED_BRIGHT * 10) + "%
");
#else
    lcd.print("indic.brt.:");
#endif
    //lcd.setCursor(15, 0);
    if (LED_BRIGHT == 11) {
#if (WEEK_LANG == 1)
      lcd.print("ABTO ");
#else
      lcd.print("Auto ");
#endif
    }
    else lcd.print(String(LED_BRIGHT * 10) + "%");
  }

  if (mode == 0) {
    lcd.clear();
    loadClock();
    drawSensors();
    if (DISPLAY_TYPE == 1) drawData();
  } else if (mode <= 10) {
    //lcd.clear();
    loadPlot();
    redrawPlot();
  }
 }
}

void redrawPlot() {
 lcd.clear();
#if (DISPLAY_TYPE == 1)
 switch (mode) {
  case 1: drawPlot(0, 3, 15, 4, CO2_MIN, CO2_MAX,
(int*)co2Hour, "c ", "hr", mode);
    break;
  case 2: drawPlot(0, 3, 15, 4, CO2_MIN, CO2_MAX, (int*)co2Day,
"c ", "da", mode);
    break;
  case 3: drawPlot(0, 3, 15, 4, HUM_MIN, HUM_MAX,
(int*)humHour, "h%", "hr", mode);
    break;
  case 4: drawPlot(0, 3, 15, 4, HUM_MIN, HUM_MAX,
(int*)humDay, "h%", "da", mode);
    break;
  case 5: drawPlot(0, 3, 15, 4, TEMP_MIN, TEMP_MAX,
(int*)tempHour, "t\337", "hr", mode);
    break;
```

```
   case 6: drawPlot(0, 3, 15, 4, TEMP_MIN, TEMP_MAX,
(int*)tempDay, "t\337", "da", mode);

     break;

   case 7: drawPlot(0, 3, 15, 4, PRESS_MIN, PRESS_MAX,
(int*)pressHour, "p ", "hr", mode);

     break;

   case 8: drawPlot(0, 3, 15, 4, PRESS_MIN, PRESS_MAX,
(int*)pressDay, "p ", "da", mode);

     break;

   case 9: drawPlot(0, 3, 15, 4, ALT_MIN, ALT_MAX, (int*)altHour,
"m ", "hr", mode);

     break;

   case 10: drawPlot(0, 3, 15, 4, ALT_MIN, ALT_MAX, (int*)altDay,
"m ", "da", mode);

     break;

  }

#else

  switch (mode) {

   case 1: drawPlot(0, 1, 12, 2, CO2_MIN, CO2_MAX,
(int*)co2Hour, "c", "h", mode);

     break;

   case 2: drawPlot(0, 1, 12, 2, CO2_MIN, CO2_MAX, (int*)co2Day,
"c", "d", mode);

     break;

   case 3: drawPlot(0, 1, 12, 2, HUM_MIN, HUM_MAX,
(int*)humHour, "h", "h", mode);

     break;

   case 4: drawPlot(0, 1, 12, 2, HUM_MIN, HUM_MAX,
(int*)humDay, "h", "d", mode);

     break;

   case 5: drawPlot(0, 1, 12, 2, TEMP_MIN, TEMP_MAX,
(int*)tempHour, "t", "h", mode);

     break;

   case 6: drawPlot(0, 1, 12, 2, TEMP_MIN, TEMP_MAX,
(int*)tempDay, "t", "d", mode);

     break;

   case 7: drawPlot(0, 1, 12, 2, PRESS_MIN, PRESS_MAX,
(int*)pressHour, "p", "h", mode);

     break;

   case 8: drawPlot(0, 1, 12, 2, PRESS_MIN, PRESS_MAX,
(int*)pressDay, "p", "d", mode);

     break;

   case 9: drawPlot(0, 1, 12, 2, ALT_MIN, ALT_MAX, (int*)altHour,
"m", "h", mode);

     break;

   case 10: drawPlot(0, 1, 12, 2, ALT_MIN, ALT_MAX, (int*)altDay,
"m", "d", mode);

     break;

  }

#endif

}


void readSensors() {

  bme.takeForcedMeasurement();

  dispTemp = bme.readTemperature();

  dispHum = bme.readHumidity();

  dispAlt = ((float)dispAlt * 1 +
bme.readAltitude(SEALEVELPRESSURE_HPA)) / 2;

  dispPres = (float)bme.readPressure() * 0.00750062;

#if (CO2_SENSOR == 1)

  dispCO2 = mhz19.getPPM();

#else

  dispCO2 = 0;

#endif

}


void drawSensors() {

#if (DISPLAY_TYPE == 1)


  if (mode0scr != 2) {

    lcd.setCursor(0, 2);

    if (bigDig) {

      if (mode0scr == 1) lcd.setCursor(15, 2);

      if (mode0scr != 1) lcd.setCursor(15, 0);

    }

    lcd.print(String(dispTemp, 1));

    lcd.write(223);

  } else {

    drawTemp(dispTemp, 0, 0);

  }


  if (mode0scr != 4) {

    lcd.setCursor(5, 2);
```

```
    if (bigDig) lcd.setCursor(15, 1);
    lcd.print(" " + String(dispHum) + "% ");
  } else {
    drawHum(dispHum, 0, 0);
  }

#if (CO2_SENSOR == 1)
  if (mode0scr != 1) {

    if (bigDig) {
      lcd.setCursor(15, 2);
      lcd.print(String(dispCO2) + "p");
    } else {
      lcd.setCursor(11, 2);
      lcd.print(String(dispCO2) + "ppm ");
    }
  } else {
    drawPPM(dispCO2, 0, 0);
  }
#endif

  if (mode0scr != 3) {
    lcd.setCursor(0, 3);
    if (bigDig && mode0scr == 0) lcd.setCursor(15, 3);
    if (bigDig && (mode0scr == 1 || mode0scr == 2))
lcd.setCursor(15, 0);
    if (bigDig && mode0scr == 4) lcd.setCursor(15, 1);
    if (!(bigDig && mode0scr == 1)) lcd.print(String(dispPres) +
"mm");
  } else {
    drawPres(dispPres, 0, 0);
  }

  if (mode0scr != 5) {
  } else {
    drawAlt(dispAlt, 0, 0);
  }

  if (!bigDig) {
    lcd.setCursor(5, 3);
    lcd.print(" rain     ");
    lcd.setCursor(11, 3);
    if (dispRain < 0) lcd.setCursor(10, 3);
    lcd.print(String(dispRain) + "%");
  }

  if (mode0scr != 0) {
    lcd.setCursor(15, 3);
    if (hrs / 10 == 0) lcd.print(" ");
    lcd.print(hrs);
    lcd.print(":");
    if (mins / 10 == 0) lcd.print("0");
    lcd.print(mins);
  } else {
    drawClock(hrs, mins, 0, 0); //, 1);
  }
#else

  if (!bigDig) {
    lcd.setCursor(0, 0);
    lcd.print(String(dispTemp, 1));
    lcd.write(223);
    lcd.setCursor(6, 0);
    lcd.print(String(dispHum) + "% ");

#if (CO2_SENSOR == 1)
    lcd.print(String(dispCO2) + "ppm");
    if (dispCO2 < 1000) lcd.print(" ");
#endif

    lcd.setCursor(0, 1);
    lcd.print(String(dispPres) + " mm  rain ");
    lcd.print(String(dispRain) + "% ");
  } else {
    switch (mode0scr) {
      case 0:
```

```
      drawClock(hrs, mins, 0, 0);
      break;
    case 1:
#if (CO2_SENSOR == 1)
      drawPPM(dispCO2, 0, 0);
#endif
      break;
    case 2:
      drawTemp(dispTemp, 2, 0);
      break;
    case 3:
      drawPres(dispPres, 2, 0);
      break;
    case 4:
      drawHum(dispHum, 0, 0);
      break;
    case 5:
      drawHum(dispAlt, 0, 0);
      break;
   }
 }
#endif
}

void plotSensorsTick() {

 if (testTimer(hourPlotTimerD, hourPlotTimer)) {
   for (byte i = 0; i < 14; i++) {
    tempHour[i] = tempHour[i + 1];
    humHour[i] = humHour[i + 1];
    pressHour[i] = pressHour[i + 1];
    altHour[i] = altHour[i + 1];
    co2Hour[i] = co2Hour[i + 1];
   }
   tempHour[14] = dispTemp;
   humHour[14] = dispHum;
   pressHour[14] = dispPres;
   altHour[14] = dispAlt;
   co2Hour[14] = dispCO2;

   if (PRESSURE) pressHour[14] = dispRain;
   else pressHour[14] = dispPres;
  }

 if (testTimer(dayPlotTimerD, dayPlotTimer)) {
   long averTemp = 0, averHum = 0, averPress = 0, averAlt = 0,
averCO2 = 0;

   for (byte i = 0; i < 15; i++) {
    averTemp += tempHour[i];
    averHum += humHour[i];
    averPress += pressHour[i];
    averAlt += altHour[i];
    averCO2 += co2Hour[i];
   }
   averTemp /= 15;
   averHum /= 15;
   averPress /= 15;
   //   averRain /= 15;
   averAlt /= 15;
   averCO2 /= 15;

   for (byte i = 0; i < 14; i++) {
    tempDay[i] = tempDay[i + 1];
    humDay[i] = humDay[i + 1];
    pressDay[i] = pressDay[i + 1];
    altDay[i] = altDay[i + 1];
    co2Day[i] = co2Day[i + 1];
   }
   tempDay[14] = averTemp;
   humDay[14] = averHum;
   pressDay[14] = averPress;
   altDay[14] = averAlt;
   co2Day[14] = averCO2;
  }

 if (testTimer(predictTimerD, predictTimer)) {
```

```
  long averPress = 0;
  for (byte i = 0; i < 10; i++) {
   bme.takeForcedMeasurement();
   averPress += bme.readPressure();
   delay(1);
  }
  averPress /= 10;

  for (byte i = 0; i < 5; i++) {
   pressure_array[i] = pressure_array[i + 1];
  }
  pressure_array[5] = averPress;
  sumX = 0;
  sumY = 0;
  sumX2 = 0;
  sumXY = 0;
  for (int i = 0; i < 6; i++) {
   //sumX += time_array[i];
   sumX += i;
   sumY += (long)pressure_array[i];
   //sumX2 += time_array[i] * time_array[i];
   sumX2 += i * i;
   //sumXY += (long)time_array[i] * pressure_array[i];
   sumXY += (long)i * pressure_array[i];
  }
  a = 0;
  a = (long)6 * sumXY;
  a = a - (long)sumX * sumY;
  a = (float)a / (6 * sumX2 - sumX * sumX);
  delta = a * 6;
  dispRain = map(delta, -250, 250, 100, -100);
 }
}

boolean dotFlag;
void clockTick() {
 dotFlag = !dotFlag;
 if (dotFlag) {
  secs++;
  if (secs > 59) {
   secs = 0;
   mins++;
   if (mins <= 59 && mode == 0) {
    drawSensors();
   }
  }
  if (mins > 59) {
   now = rtc.now();
   secs = now.second();
   mins = now.minute();
   hrs = now.hour();
   if (mode == 0) drawSensors();
   if (hrs > 23) hrs = 0;
   if (mode == 0 && DISPLAY_TYPE) drawData();
  }
  if ((DISP_MODE != 0 && mode == 0) && DISPLAY_TYPE == 1 &&
!bigDig) {
   lcd.setCursor(15, 1);
   if (secs < 10) lcd.print(" ");
   lcd.print(secs);
  }
 }

 if (mode == 0) {

  if (!bigDig && powerStatus != 255 && DISPLAY_TYPE == 1) {

   if (analogRead(A1) > 900 || analogRead(A0) < 300 ||
(analogRead(A1) < 300 && analogRead(A0) < 300)) powerStatus =
0;
   else powerStatus = (constrain((int)analogRead(A0) * 5.2 /
1023.0, 3.0, 4.2) - 3.0) / ((4.2 - 3.0) / 6.0) + 1;

   if (powerStatus) {
    for (byte i = 2; i <= 6; i++) {
     if ((7 - powerStatus) < i) DC[i] = 0b11111;
     else DC[i] = 0b10001;
    }
```

```
      lcd.createChar(6, DC);
    } else lcd.createChar(6, AC);

    if (mode0scr != 1) lcd.setCursor(19, 2);
    else lcd.setCursor(19, 0);
    if (!dotFlag && powerStatus == 1) lcd.write(32);
    else lcd.write(6);
  }

  byte code;
  if (dotFlag) code = 165;
  else code = 32;
  if (mode0scr == 0 && (bigDig && DISPLAY_TYPE == 0 ||
DISPLAY_TYPE == 1)) {
    if (bigDig && DISPLAY_TYPE == 1) lcd.setCursor(7, 2);
    else lcd.setCursor(7, 0);
    lcd.write(code);
    lcd.setCursor(7, 1);
    lcd.write(code);
  }
  else {
#if (DISPLAY_TYPE == 1)
    if (code == 165) code = 58;
    lcd.setCursor(17, 3);
    lcd.write(code);
#endif
  }
 }

 if ((dispCO2 >= blinkLEDCO2 && LEDType == 0 || dispHum <=
blinkLEDHum && LEDType == 1 || dispTemp >= blinkLEDTemp &&
LEDType == 2) && !dotFlag) setLEDcolor(0);
 else setLED();
}

boolean testTimer(unsigned long & dataTimer, unsigned long
setTimer) {
 if (millis() - dataTimer >= setTimer) {
  dataTimer = millis();
  return true;
 } else {
  return false;
 }
}
```

```
#define RESET_CLOCK 0
#define SENS_TIME 10000
#define LED_MODE 0
#define SEALEVELPRESSURE_HPA (1013.25)

byte LED_BRIGHT = 10;
byte LCD_BRIGHT = 10;
byte powerStatus = 0;
#define BRIGHT_CONTROL 1
#define BRIGHT_THRESHOLD 350
#define LED_BRIGHT_MAX 255
#define LED_BRIGHT_MIN 10
#define LCD_BRIGHT_MAX 255
#define LCD_BRIGHT_MIN 10

#define DISP_MODE 1
#define WEEK_LANG 1
#define DEBUG 0
#define PRESSURE 0
#define CO2_SENSOR 0
#define DISPLAY_TYPE 1
#define DISPLAY_ADDR 0x27
```

```
#define normCO2 800
#define maxCO2 1200
#define blinkLEDCO2 1500

#define minTemp 21
#define normTemp 26
#define maxTemp 28
#define blinkLEDTemp 35

#define maxHum 90
#define normHum 30
#define minHum 20
#define blinkLEDHum 15

#define normPress 733
#define minPress 728

#define minRain -50
#define normRain -20
#define maxRain 50


byte LEDType = 0;

#include <EEPROM.h>

int MAX_ONDATA = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512
+ 1024 + 2048;
int VIS_ONDATA = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512 +
1024 + 2048;


#define TEMP_MIN 15
#define TEMP_MAX 35
#define HUM_MIN 0
#define HUM_MAX 100
#define PRESS_MIN 720
#define PRESS_MAX 760
//#define PRESS_MIN -100
//#define PRESS_MAX 100
#define CO2_MIN 400
#define CO2_MAX 2000
#define ALT_MIN 0
#define ALT_MAX 1000

#define BACKLIGHT 10
#define PHOTO A3

#define MHZ_RX 2
#define MHZ_TX 3

#define LED_COM 7
#define LED_R 9
#define LED_G 6
#define LED_B 5
#define BTN_PIN 4

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#if (DISPLAY_TYPE == 1)
LiquidCrystal_I2C lcd(DISPLAY_ADDR, 20, 4);
#else
LiquidCrystal_I2C lcd(DISPLAY_ADDR, 16, 2);
#endif

#include "RTClib.h"
RTC_DS3231 rtc;
DateTime now;

#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme;

#if (CO2_SENSOR == 1)
```

```
#include <MHZ19_uart.h>
MHZ19_uart mhz19;
#endif

unsigned long sensorsTimer = SENS_TIME;
unsigned long drawSensorsTimer = SENS_TIME;
unsigned long clockTimer = 500;

#if (DISPLAY_TYPE == 1)
unsigned long hourPlotTimer = ((long)4 * 60 * 1000);
unsigned long dayPlotTimer = ((long)1.6 * 60 * 60 * 1000);
#else
unsigned long hourPlotTimer = ((long)5 * 60 * 1000);
unsigned long dayPlotTimer = ((long)2 * 60 * 60 * 1000);
#endif

unsigned long predictTimer = ((long)10 * 60 * 1000);
unsigned long plotTimer = hourPlotTimer;
unsigned long brightTimer = (2000);

unsigned long sensorsTimerD = 0;
unsigned long drawSensorsTimerD = 0;
unsigned long clockTimerD = 0;
unsigned long hourPlotTimerD = 0;
unsigned long dayPlotTimerD = 0;
unsigned long plotTimerD = 0;
unsigned long predictTimerD = 0;
unsigned long brightTimerD = 0;

#include "GyverButton.h"
GButton button(BTN_PIN, LOW_PULL, NORM_OPEN);

int8_t hrs, mins, secs;
byte mode = 0;

byte podMode = 1;
byte mode0scr = 0;
boolean bigDig = false;

float dispTemp;
byte dispHum;
int dispPres;
int dispCO2 = -1;
int dispRain;
float dispAlt;

int tempHour[15], tempDay[15];
int humHour[15], humDay[15];
int pressHour[15], pressDay[15];
int rainHour[15], rainDay[15];
int co2Hour[15], co2Day[15];
int altHour[15], altDay[15];
int delta;
uint32_t pressure_array[6];
uint32_t sumX, sumY, sumX2, sumXY;
float a, b;

byte rowS[8] = {0b00000, 0b00000, 0b00000, 0b00000, 0b10001, 0b01010, 0b00100, 0b00000};
byte row7[8] = {0b00000, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111};
byte row6[8] = {0b00000, 0b00000, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111};
byte row5[8] = {0b00000, 0b00000, 0b00000, 0b11111, 0b11111, 0b11111, 0b11111, 0b11111};
byte row4[8] = {0b00000, 0b00000, 0b00000, 0b00000, 0b11111, 0b11111, 0b11111, 0b11111};
byte row3[8] = {0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b11111, 0b11111, 0b11111};
byte row2[8] = {0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b11111, 0b11111};
byte row1[8] = {0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b00000, 0b11111};
```

```
uint8_t UB[8] = {0b11111, 0b11111, 0b11111, 0b00000,
0b00000, 0b00000, 0b00000, 0b00000};

uint8_t UMB[8] = {0b11111, 0b11111, 0b11111, 0b00000,
0b00000, 0b00000, 0b11111, 0b11111};

uint8_t LMB[8] = {0b11111, 0b00000, 0b00000, 0b00000,
0b00000, 0b11111, 0b11111, 0b11111};

uint8_t LM2[8] = {0b11111, 0b00000, 0b00000, 0b00000,
0b00000, 0b00000, 0b00000, 0b00000};

uint8_t UT[8] = {0b11111, 0b11111, 0b11111, 0b11111,
0b11111, 0b00000, 0b00000, 0b00000};


uint8_t KU[8] = {0b00000, 0b00000, 0b00000, 0b00001,
0b00010, 0b00100, 0b01000, 0b10000};

uint8_t KD[8] = {0b00001, 0b00010, 0b00100, 0b01000,
0b10000, 0b00000, 0b00000, 0b00000};


uint8_t PP[8] = {0b11111, 0b10001, 0b10001, 0b10001,
0b10001, 0b10001, 0b10001, 0b00000};

uint8_t BB[8] = {0b11111, 0b10000, 0b10000, 0b11111,
0b10001, 0b10001, 0b11111, 0b00000};

uint8_t CH[8] = {0b10001, 0b10001, 0b10001, 0b01111,
0b00001, 0b00001, 0b00001, 0b00000};

uint8_t II[8] = {0b10001, 0b10001, 0b10011, 0b10101, 0b11001,
0b10001, 0b10001, 0b00000};

uint8_t BM[8] = {0b10000, 0b10000, 0b10000, 0b11110,
0b10001, 0b10001, 0b11110, 0b00000};

uint8_t IY[8] = {0b01100, 0b00001, 0b10011, 0b10101,
0b11001, 0b10001, 0b10001, 0b00000};

uint8_t DD[8] = {0b01110, 0b01010, 0b01010, 0b01010,
0b01010, 0b01010, 0b11111, 0b10001};

uint8_t AA[8] = {0b11100, 0b00010, 0b00001, 0b00111,
0b00001, 0b00010, 0b11100, 0b00000};

uint8_t IA[8] = {0b01111, 0b10001, 0b10001, 0b01111,
0b00101, 0b01001, 0b10001, 0b00000};

uint8_t YY[8] = {0b10001, 0b10001, 0b10001, 0b11101,
0b10011, 0b10011, 0b11101, 0b00000};

uint8_t GG[8] = {0b11110, 0b10000, 0b10000, 0b10000,
0b10000, 0b10000, 0b10000, 0b00000};

uint8_t FF[8] = {0b00100, 0b01110, 0b10101, 0b10101,
0b10101, 0b01110, 0b00100, 0b00000};

uint8_t LL[8] = {0b01111, 0b01001, 0b01001, 0b01001,
0b01001, 0b01001, 0b10001, 0b00000};

uint8_t ZZ[8] = {0b10101, 0b10101, 0b10101, 0b01110,
0b10101, 0b10101, 0b10101, 0b00000};



uint8_t AC[8] = {0b01010, 0b01010, 0b11111, 0b11111,
0b01110, 0b00100, 0b00100, 0b00011};

uint8_t DC[8] = {0b01110, 0b11111, 0b11111, 0b11111,
0b11111, 0b11111, 0b11111, 0b11111};


void digSeg(byte x, byte y, byte z1, byte z2, byte z3, byte z4, byte
z5, byte z6) {
 lcd.setCursor(x, y);
 lcd.write(z1); lcd.write(z2); lcd.write(z3);
 if (x <= 11) lcd.print(" ");
 lcd.setCursor(x, y + 1);
 lcd.write(z4); lcd.write(z5); lcd.write(z6);
 if (x <= 11) lcd.print(" ");
}


void drawDig(byte dig, byte x, byte y) {
 if (bigDig && DISPLAY_TYPE == 1) {
  switch (dig) {
   case 0:
    digSeg(x, y, 255, 0, 255, 255, 32, 255);
    digSeg(x, y + 2, 255, 32, 255, 255, 3, 255);
    break;
   case 1:
    digSeg(x, y, 32, 255, 32, 32, 255, 32);
    digSeg(x, y + 2, 32, 255, 32, 32, 255, 32);
    break;
   case 2:
    digSeg(x, y, 0, 0, 255, 1, 1, 255);
    digSeg(x, y + 2, 255, 2, 2, 255, 3, 3);
    break;
   case 3:
    digSeg(x, y, 0, 0, 255, 1, 1, 255);
    digSeg(x, y + 2, 2, 2, 255, 3, 3, 255);
    break;
   case 4:
    digSeg(x, y, 255, 32, 255, 255, 1, 255);
    digSeg(x, y + 2, 2, 2, 255, 32, 32, 255);
    break;
   case 5:
    digSeg(x, y, 255, 0, 0, 255, 1, 1);
    digSeg(x, y + 2, 2, 2, 255, 3, 3, 255);
```

```
    break;
   case 6:
    digSeg(x, y, 255, 0, 0, 255, 1, 1);
    digSeg(x, y + 2, 255, 2, 255, 255, 3, 255);
    break;
   case 7:
    digSeg(x, y, 0, 0, 255, 32, 32, 255);
    digSeg(x, y + 2, 32, 255, 32, 32, 255, 32);
    break;
   case 8:
    digSeg(x, y, 255, 0, 255, 255, 1, 255);
    digSeg(x, y + 2, 255, 2, 255, 255, 3, 255);
    break;
   case 9:
    digSeg(x, y, 255, 0, 255, 255, 1, 255);
    digSeg(x, y + 2, 2, 2, 255, 3, 3, 255);
    break;
   case 10:
    digSeg(x, y, 32, 32, 32, 32, 32, 32);
    digSeg(x, y + 2, 32, 32, 32, 32, 32, 32);
    break;
  }
 }
 else {
  switch (dig) {
   case 0:
    digSeg(x, y, 255, 1, 255, 255, 2, 255);
    break;
   case 1:
    digSeg(x, y, 32, 255, 32, 32, 255, 32);
    break;
   case 2:
    digSeg(x, y, 3, 3, 255, 255, 4, 4);
    break;
   case 3:
    digSeg(x, y, 3, 3, 255, 4, 4, 255);
    break;
   case 4:
    digSeg(x, y, 255, 0, 255, 5, 5, 255);
    break;
   case 5:
    digSeg(x, y, 255, 3, 3, 4, 4, 255);
    break;
   case 6:
    digSeg(x, y, 255, 3, 3, 255, 4, 255);
    break;
   case 7:
    digSeg(x, y, 1, 1, 255, 32, 255, 32);
    break;
   case 8:
    digSeg(x, y, 255, 3, 255, 255, 4, 255);
    break;
   case 9:
    digSeg(x, y, 255, 3, 255, 4, 4, 255);
    break;
   case 10:
    digSeg(x, y, 32, 32, 32, 32, 32, 32);
    break;
  }
 }
}

void drawPPM(int dispCO2, byte x, byte y) {
 if (dispCO2 / 1000 == 0) drawDig(10, x, y);
 else drawDig(dispCO2 / 1000, x, y);
 drawDig((dispCO2 % 1000) / 100, x + 4, y);
 drawDig((dispCO2 % 100) / 10, x + 8, y);
 drawDig(dispCO2 % 10 , x + 12, y);
 lcd.setCursor(15, 0);
#if (DISPLAY_TYPE == 1)
 lcd.print("ppm");
#else
 lcd.print("p");
#endif
}
```

```
void drawPres(int dispPres, byte x, byte y) {
  drawDig((dispPres % 1000) / 100, x , y);
  drawDig((dispPres % 100) / 10, x + 4, y);
  drawDig(dispPres % 10 , x + 8, y);
  lcd.setCursor(x + 11, 1);
  if (bigDig) lcd.setCursor(x + 11, 3);
  lcd.print("mm");
}

void drawAlt(float dispAlt, byte x, byte y) {
  if (dispAlt >= 1000) {
    drawDig((int(dispAlt) % 10000) / 1000, x , y);
    x += 4;
  }
  drawDig((int(dispAlt) % 1000) / 100, x , y);
  drawDig((int(dispAlt) % 100) / 10, x + 4, y);
  drawDig(int(dispAlt) % 10 , x + 8, y);
  if (dispAlt < 1000) {
    //   drawDig((int(dispAlt * 10.0)) % 10 , x + 12, y);
    lcd.setCursor(x + 12, y + 1 + (bigDig && DISPLAY_TYPE) * 2);
    lcd.print((int(dispAlt * 10.0)) % 10);
    if (bigDig && DISPLAY_TYPE == 1) lcd.setCursor(x + 11, y + 3);
    else lcd.setCursor(x + 11, y + 1);
    lcd.print(".");
    x -= 1;
  }  else {
    x -= 4;
  }
  if (bigDig && DISPLAY_TYPE == 1) lcd.setCursor(x + 14, 3);
  else lcd.setCursor(x + 14, 1);
  lcd.print("m");
}

void drawTemp(float dispTemp, byte x, byte y) {
  if (dispTemp / 10 == 0) drawDig(10, x, y);
  else drawDig(dispTemp / 10, x, y);
  drawDig(int(dispTemp) % 10, x + 4, y);
  drawDig(int(dispTemp * 10.0) % 10, x + 9, y);
  if (bigDig && DISPLAY_TYPE == 1) {
    lcd.setCursor(x + 7, y + 3);
    lcd.write(1);
  }
  else {
    lcd.setCursor(x + 7, y + 1);
    lcd.write(0B10100001);
  }
  lcd.setCursor(x + 13, y);
  lcd.write(223);
}

void drawHum(int dispHum, byte x, byte y) {
  if (dispHum / 100 == 0) drawDig(10, x, y);
  else drawDig(dispHum / 100, x, y);
  if ((dispHum % 100) / 10 == 0) drawDig(0, x + 4, y);
  else drawDig(dispHum / 10, x + 4, y);
  drawDig(int(dispHum) % 10, x + 8, y);
  if (bigDig && DISPLAY_TYPE == 1) {
    lcd.setCursor(x + 12, y + 1);
    lcd.print("\245\4");
    lcd.setCursor(x + 12, y + 2);
    lcd.print("\5\245");
  }
  else {
    lcd.setCursor(x + 12, y + 1);
    lcd.print("%");
  }
}

void drawClock(byte hours, byte minutes, byte x, byte y) {
  if (hours > 23 || minutes > 59) return;
  if (hours / 10 == 0) drawDig(10, x, y);
  else drawDig(hours / 10, x, y);
  drawDig(hours % 10, x + 4, y);

  drawDig(minutes / 10, x + 8, y);
```

```
  drawDig(minutes % 10, x + 12, y);
}

#if (WEEK_LANG == 0)
static const char *dayNames[]  = {
  "Su",
  "Mo",
  "Tu",
  "We",
  "Th",
  "Fr",
  "Sa",
};
#else
static const char *dayNames[]  = {
  "BC",
  "\7H",
  "BT",
  "CP",
  "\7T",
  "\7T",
  "C\7",
};
#endif

void drawData() {
  int Y = 0;
  if (DISPLAY_TYPE == 1 && mode0scr == 1) Y = 2;
  if (!bigDig) {
    lcd.setCursor(15, 0 + Y);
    if (now.day() < 10) lcd.print(0);
    lcd.print(now.day());
    lcd.print(".");
    if (now.month() < 10) lcd.print(0);
    lcd.print(now.month());

    if (DISP_MODE == 0) {
      lcd.setCursor(16, 1);
      lcd.print(now.year());
    } else {
      loadClock();
      lcd.setCursor(18, 1);
      int dayofweek = now.dayOfTheWeek();
      lcd.print(dayNames[dayofweek]);
      // if (hrs == 0 && mins == 0 && secs <= 1) loadClock();
    }
  }
}

void drawPlot(byte pos, byte row, byte width, byte height, int
min_val, int max_val, int *plot_array, String label1, String label2,
int stretch) {
  int max_value = -32000;
  int min_value = 32000;

  for (byte i = 0; i < 15; i++) {
    max_value = max(plot_array[i] , max_value);
    min_value = min(plot_array[i] , min_value);
  }

  lcd.setCursor(15, 0);
  if ((MAX_ONDATA & (1 << (stretch - 1))) > 0) {
    max_val = max_value;
    min_val = min_value;
#if (DISPLAY_TYPE == 1)
    lcd.write(0b01011110);
    lcd.setCursor(15, 3);
    lcd.write(0);
#endif
  }  else {
#if (DISPLAY_TYPE == 1)
    lcd.write(0);
    lcd.setCursor(15, 3);
    lcd.write(0b01011110);
#endif
  }
```

```
  if (min_val >= max_val) max_val = min_val + 1;
#if (DISPLAY_TYPE == 1)
  lcd.setCursor(15, 1); lcd.write(0b01111100);
  lcd.setCursor(15, 2); lcd.write(0b01111100);

  //Serial.println(max_val);Serial.println(min_val);

  lcd.setCursor(16, 0); lcd.print(max_value);
  lcd.setCursor(16, 1); lcd.print(label1); lcd.print(label2);
  lcd.setCursor(16, 2); lcd.print(plot_array[14]);
  lcd.setCursor(16, 3); lcd.print(min_value);
#else
  lcd.setCursor(12, 0); lcd.print(label1);
  lcd.setCursor(13, 0); lcd.print(max_value);
  lcd.setCursor(12, 1); lcd.print(label2);
  lcd.setCursor(13, 1); lcd.print(min_value);
#endif
  for (byte i = 0; i < width; i++) {
    int fill_val = plot_array[i];
    fill_val = constrain(fill_val, min_val, max_val);
    byte infill, fract;

    if ((plot_array[i]) > min_val)
      infill = floor((float)(plot_array[i] - min_val) / (max_val - min_val)
* height * 10);
    else infill = 0;
    fract = (float)(infill % 10) * 8 / 10;
    infill = infill / 10;

    for (byte n = 0; n < height; n++) {
      if (n < infill && infill > 0) {
        lcd.setCursor(i, (row - n));
        lcd.write(255);
      }
      if (n >= infill) {
        lcd.setCursor(i, (row - n));
        if (n == 0 && fract == 0) fract++;
        if (fract > 0) lcd.write(fract);
        else lcd.write(16);
        for (byte k = n + 1; k < height; k++) {
          lcd.setCursor(i, (row - k));
          lcd.write(16);
        }
        break;
      }
    }
  }
}

void loadClock() {
  if (bigDig && (DISPLAY_TYPE == 1)) {
    lcd.createChar(0, UT);
    lcd.createChar(1, row3);
    lcd.createChar(2, UB);
    lcd.createChar(3, row5);
    lcd.createChar(4, KU);
    lcd.createChar(5, KD);
  }
  else {
    lcd.createChar(0, row2);
    lcd.createChar(1, UB);
    lcd.createChar(2, row3);
    lcd.createChar(3, UMB);
    lcd.createChar(4, LMB);
    lcd.createChar(5, LM2);
  }

  if (now.dayOfTheWeek() == 4)  {
    lcd.createChar(7, CH);
  } else if (now.dayOfTheWeek() == 6) {
    lcd.createChar(7, BB);
  } else {
    lcd.createChar(7, PP);
  }
}
```

```
void loadPlot() {
  lcd.createChar(0, rowS);
  lcd.createChar(1, row1);
  lcd.createChar(2, row2);
  lcd.createChar(3, row3);
  lcd.createChar(4, row4);
  lcd.createChar(5, row5);
  lcd.createChar(6, row6);
  lcd.createChar(7, row7);
}

#if (LED_MODE == 0)
byte LED_ON = (LED_BRIGHT_MAX);
byte LED_OFF = (0);
#else
byte LED_ON = (255 - LED_BRIGHT_MAX);
byte LED_OFF = (255);
#endif

void setLEDcolor(byte color) {
  analogWrite(LED_R, LED_ON + LED_ON * ((LED_MODE << 1) - 1) *
(3 - (color & 3)) / 3);
  analogWrite(LED_G, LED_ON + LED_ON * ((LED_MODE << 1) - 1)
* (3 - ((color & 12) >> 2)) / 3);
  analogWrite(LED_B, LED_ON + LED_ON * ((LED_MODE << 1) - 1) *
(3 - ((color & 48) >> 4)) / 3);
}

void setLED() {

  if (LED_BRIGHT < 11) {
    LED_ON = 255 / 100 * LED_BRIGHT * LED_BRIGHT;
  } else {
    checkBrightness();
  }
  if (LED_MODE != 0) LED_ON = 255 - LED_ON;

  if ((dispCO2 >= maxCO2) && LEDType == 0 || (dispHum <=
minHum) && LEDType == 1 || (dispTemp >= maxTemp) &&
LEDType == 2 || (dispRain <= minRain) && LEDType == 3 ||
(dispPres <= minPress) && LEDType == 4) setLEDcolor(3);
  else if ((dispCO2 >= normCO2) && LEDType == 0 || (dispHum <=
normHum) && LEDType == 1 || (dispTemp >= normTemp) &&
LEDType == 2 || (dispRain <= normRain) && LEDType == 3 ||
(dispPres <= normPress) && LEDType == 4) setLEDcolor(3 + 8);
  else if (LEDType == 0 || (dispHum <= maxHum) && LEDType == 1
|| (dispTemp >= minTemp) && LEDType == 2 || (dispRain <=
maxRain) && LEDType == 3 || LEDType == 4) setLEDcolor(12);
  else setLEDcolor(48);
}

void setup() {
  Serial.begin(9600);

  pinMode(BACKLIGHT, OUTPUT);
  pinMode(LED_COM, OUTPUT);
  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
  setLEDcolor(0);

  digitalWrite(LED_COM, LED_MODE);
  analogWrite(BACKLIGHT, LCD_BRIGHT_MAX);

  if (EEPROM.read(0) == 122) {
    MAX_ONDATA = EEPROM.read(2);
    MAX_ONDATA += (long)(EEPROM.read(3) << 8);
    VIS_ONDATA = EEPROM.read(4);
    VIS_ONDATA += (long)(EEPROM.read(5) << 8);
    mode0scr = EEPROM.read(6);
    bigDig = EEPROM.read(7);
    LED_BRIGHT = EEPROM.read(8);
    LCD_BRIGHT = EEPROM.read(9);
    LEDType = EEPROM.read(10);
  }

  lcd.init();
  lcd.backlight();
  lcd.clear();
```

```
#if (DEBUG == 1 && DISPLAY_TYPE == 1)
  boolean status = true;

  setLEDcolor(3);

#if (CO2_SENSOR == 1)
  lcd.setCursor(0, 0);
  lcd.print(F("MHZ-19... "));
  Serial.print(F("MHZ-19... "));
  mhz19.begin(MHZ_TX, MHZ_RX);
  mhz19.setAutoCalibration(false);
  mhz19.getStatus();
  delay(500);
  if (mhz19.getStatus() == 0) {
    lcd.print(F("OK"));
    Serial.println(F("OK"));
  } else {
    lcd.print(F("ERROR"));
    Serial.println(F("ERROR"));
    status = false;
  }
#endif

  setLEDcolor(3 + 12);
  lcd.setCursor(0, 1);
  lcd.print(F("RTC... "));
  Serial.print(F("RTC... "));
  delay(50);
  if (rtc.begin()) {
    lcd.print(F("OK"));
    Serial.println(F("OK"));
  } else {
    lcd.print(F("ERROR"));
    Serial.println(F("ERROR"));
    status = false;
  }

  setLEDcolor(12);
  lcd.setCursor(0, 2);
  lcd.print(F("BME280... "));
  Serial.print(F("BME280... "));
  delay(50);
  if (bme.begin(&Wire)) {
    lcd.print(F("OK"));
    Serial.println(F("OK"));
  } else {
    lcd.print(F("ERROR"));
    Serial.println(F("ERROR"));
    status = false;
  }

  setLEDcolor(0);
  lcd.setCursor(0, 3);
  if (status) {
    lcd.print(F("All good"));
    Serial.println(F("All good"));
  } else {
    lcd.print(F("Check wires!"));
    Serial.println(F("Check wires!"));
  }

  for (byte i = 1; i < 20; i++) {
    lcd.setCursor(14, 1);
    lcd.print("P:    ");
    lcd.setCursor(16, 1);
    lcd.print(analogRead(PHOTO), 1);
    Serial.println(analogRead(PHOTO));
    delay(250);
  }
#else

#if (CO2_SENSOR == 1)
  mhz19.begin(MHZ_TX, MHZ_RX);
  mhz19.setAutoCalibration(false);
#endif
```

```
  rtc.begin();
  bme.begin(&Wire);
#endif

  bme.setSampling(Adafruit_BME280::MODE_FORCED,
                  Adafruit_BME280::SAMPLING_X1,
                  Adafruit_BME280::SAMPLING_X1,
                  Adafruit_BME280::SAMPLING_X1,
                  Adafruit_BME280::FILTER_OFF);

  if (RESET_CLOCK || rtc.lostPower())
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  lcd.clear();
  now = rtc.now();
  secs = now.second();
  mins = now.minute();
  hrs = now.hour();

  bme.takeForcedMeasurement();
  uint32_t Pressure = bme.readPressure();
  for (byte i = 0; i < 6; i++) {
    pressure_array[i] = Pressure;
  }

  dispAlt = (float)bme.readAltitude(SEALEVELPRESSURE_HPA);



  readSensors();
  for (byte i = 0; i < 15; i++) {
    tempHour[i] = dispTemp;
    tempDay[i] = dispTemp;
    humHour[i] = dispHum;
    humDay[i] = dispHum;
    altHour[i] = dispAlt;
    altDay[i] = dispAlt;
    if (PRESSURE) {
      pressHour[i] = 0;
      pressDay[i] = 0;
    } else {
      pressHour[i] = dispPres;
      pressDay[i] = dispPres;
    }

  }

  if (DISPLAY_TYPE == 1) drawData();
  loadClock();
  drawSensors();
}

void loop() {
  if (testTimer(brightTimerD, brightTimer)) checkBrightness();
  if (testTimer(sensorsTimerD, sensorsTimer)) readSensors();
  Serial.println(dispTemp);

  if (testTimer(clockTimerD, clockTimer)) clockTick();
  plotSensorsTick();
  modesTick();
  if (mode == 0) {
    if (testTimer(drawSensorsTimerD, drawSensorsTimer))
drawSensors();
  } else {
    if (testTimer(plotTimerD, plotTimer)) redrawPlot();
  }
}
```