МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра електроніки і комп'ютерної техніки

«До захисту допущено»

Завідувач кафедри ЕКТ

_____ Анатолій ОПАНАСЮК

(підпис) (Ім’я та ПРІЗВИЩЕ)

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

зі спеціальності 171 «Електроніка»

освітньо-професійної програми «Електронні системи та компоненти»

на тему:

Метеостанція на базі Arduino Nano з виводом даних з датчиків на телефон.

Здобувача групи ЕС-01 _____ Наумова Андрій

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ _____

(підпис) (Ім’я та ПРІЗВИЩЕ)

Керівник, завідувач кафедри, д.ф -м.н., професор Опанасюк Анатолій _____

(підпис)

Суми – 2024

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет електроніки та інформаційних технологій

Кафедра електроніки і комп'ютерної техніки

Напрям підготовки 171 Електроніка

Освітня програма Електронні системи та компоненти

ЗАТВЕРДЖУЮ

Зав. кафедрою Опанасюк А. С.

"___" _____ 2024 р.

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра

1. Тема роботи _____

затверджена наказом по університету "13" березня 2024 р. № 0256-VI.

2. Термін здачі студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що належить розробити) 1) Огляд літератури та постановка задачі роботи. 2) Розробка структурної схеми проектованого електронного пристрою. 3) Розробка алгоритму роботи проектованого електронного пристрою. 4) Розробка функціональної схеми проектованого електронного пристрою. 5) Розробка принципових схем блоків проектованого електронного пристрою. 6) Розробка програмного забезпечення проектованого електронного пристрою (при необхідності).

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

1) Схема електрична структурна. 2) Схема алгоритму. 3) Схема електрична функціональна. 4) Схема електрична принципова.

6. Дата видачі завдання _____

8. Керівник роботи _____

9. Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту	Термін виконання етапів роботи	Примітки
1	Огляд літератури та постановка завдання проектування	06.05.24 – 09.05.24	
2	Розробка структурної схеми проектованого електронного пристрою	10.05.24 – 13.05.24	
3	Розробка алгоритму роботи проектованого електронного пристрою	14.05.24 – 16.05.24	
4	Розробка функціональної схеми проектованого електронного пристрою	17.05.24 – 22.05.24	
5	Розробка принципових схем блоків проектованого електронного пристрою	23.05.24 – 30.05.24	
6	Розробка програмного забезпечення проектованого електронного пристрою	31.05.24-04.06.24	
7	Оформлення пояснювальної записки	05.06.24 – 07.06.24	
8	Оформлення графічного матеріалу	08.06.24 – 09.06.24	
9	Представлення роботи керівнику і отримання відгуку	10.06.24	
10	Представлення роботи кафедрі для отримання рецензії	10.06.24	

Студент _____

Керівник роботи _____

«___» _____ 2024 р.

РЕФЕРАТ

Пояснювальна записка містить 58 сторінок, 17 рисунків, 11 джерел, складається з вступу, чотирьох розділів, висновків, списку літератури. Графічна частина представлена структурною, блок-схемою алгоритму функціонування, принциповою схемою.

Розроблений пристрій характеризується великою чистотою спектра, гнучким у розробці та універсальним приладом який буде корисним для людини.

Зміст

ВСТУП

Проект "Метеостанція на базі Arduino Nano з виводом даних з датчиків на телефон" є надзвичайно актуальним у сучасному світі, де автоматизація та інтеграція інтелектуальних систем стає невід'ємною частиною повсякденного життя. В умовах постійного зростання інтересу до моніторингу кліматичних умов, як у приватних будинках, так і в комерційних приміщеннях, створення доступних та ефективних засобів для цього завдання є вкрай важливим. Цей проект спрямований на розробку простої у використанні та доступної системи для зчитування та аналізу кліматичних параметрів, таких як температура, вологість та атмосферний тиск, з подальшим виведенням цих даних на мобільний пристрій через USB-з'єднання.

Інтерес до метеостанцій пояснюється їх багатофункціональністю та широким спектром застосувань. Вони можуть бути використані не лише для спостереження за погодними умовами на вулиці, але й для контролю мікроклімату всередині приміщень. Наприклад, у теплицях для забезпечення оптимальних умов вирощування рослин, у складських приміщеннях для збереження продукції, або ж у житлових будинках для покращення комфортних умов проживання. Крім того, такі системи можуть бути корисними для людей, що займаються метеорологічними дослідженнями або екологічним моніторингом.

Особливістю даного проекту є використання платформи Arduino Nano у поєднанні з датчиками DHT11 і BMP180. Arduino Nano, як мініатюрний та потужний контролер, ідеально підходить для реалізації такого проекту завдяки своїй компактності та широкому функціоналу. Датчик DHT11 забезпечує зчитування температури та вологості повітря, тоді як BMP180

вимірює атмосферний тиск з високою точністю. Ці датчики є відносно дешевими та легкими у використанні, що робить проект доступним навіть для початківців у сфері електроніки та програмування.

Важливою частиною проекту є розробка мобільного додатку для платформи Android, який дозволяє отримувати та відображати дані з метеостанції у реальному часі. Використання USB-з'єднання для передачі даних між Arduino та смартфоном забезпечує стабільний та швидкий обмін інформацією, що є критично важливим для точного моніторингу кліматичних умов. Такий підхід також спрощує процес налаштування та використання системи, оскільки не вимагає наявності додаткових модулів бездротового зв'язку.

Процес розробки даного проекту включає кілька ключових етапів: вибір та підключення апаратних компонентів, написання програмного забезпечення для зчитування даних з датчиків та їх передачі на мобільний пристрій, розробка інтерфейсу мобільного додатку, інтеграція та тестування системи. Кожен з цих етапів є важливим для досягнення кінцевої мети - створення надійної та функціональної метеостанції.

Проект "Метеостанція на базі Arduino Nano з виводом даних з датчиків на телефон" є прикладом того, як сучасні технології можуть бути використані для покращення якості життя та забезпечення комфортних умов у різних сферах діяльності. Він демонструє можливості простих та доступних засобів для збору та аналізу кліматичних даних, що є важливим кроком у напрямку розвитку інтелектуальних систем моніторингу.

Завдання до кваліфікаційної роботи

Мета проекту

Розробка пристрою для збору та відображення даних про температуру, вологість та тиск повітря в приміщенні на мобільний телефон за допомогою платформи Arduino Nano та передачі даних через USB кабель.

Основні завдання проекту

1. **Збір компонентів та збирання метеостанції:**

    - Закупівля Arduino Nano, датчика температури і вологості DHT11, датчика тиску повітря BMP180, а також необхідних дротів і з'єднувачів.
    - Збирання метеостанції відповідно до схеми з'єднання, з урахуванням використання USB кабеля для передачі даних.

2. **Написання програмного коду для Arduino Nano:**

    - Програмування зчитування даних з датчиків DHT11 та BMP180.
    - Налаштування передачі даних на мобільний пристрій через USB кабель.

3. **Розробка мобільного додатка:**

    - Створення додатка для мобільного телефону для отримання даних з метеостанції.
    - Відображення отриманих даних у зручному форматі.

4. **Тестування та налагодження системи:**

    - Перевірка коректності зчитування даних з датчиків.

    - Впевнення в стабільності підключення до мобільного пристрою через USB кабель.
    - Виправлення помилок та уточнення програмного коду та мобільного додатка для оптимальної роботи системи.

5. **Встановлення метеостанції та тестування:**

    - Монтаж метеостанції в потрібному місці для збору даних.
    - Тестування роботи метеостанції в реальних умовах для перевірки її ефективності та надійності.

Документування проекту та його результатів:

- Підготовка документації, що описує всі етапи проекту, включаючи схеми з'єднання, програмний код та інструкції щодо використання метеостанції.
- Детальний опис результатів тестування та рекомендації щодо подальшого вдосконалення системи.

Опис проекту

Метою проекту є створення функціональної метеостанції, яка буде вимірювати температуру, вологість та тиск повітря, а зібрані дані передаватимуться на мобільний телефон через USB кабель. Це забезпечить надійне з'єднання без необхідності використання додаткових бездротових модулів.

Першим етапом є збір всіх необхідних компонентів, таких як Arduino Nano, датчики DHT11 та BMP180, дроти та з'єднувачі. Після цього метеостанцію потрібно зібрати відповідно до схеми з'єднання.

Наступним кроком є написання програмного коду для Arduino Nano. Код має забезпечувати зчитування даних з датчиків та передачу цих даних на мобільний пристрій через USB кабель. Це дозволить користувачам отримувати актуальну інформацію про стан навколишнього середовища безпосередньо на свій телефон.

Для зручності користувачів створюється мобільний додаток, який буде відображати отримані дані у зручному форматі. Додаток повинен бути простим у використанні та надійним.

Після завершення розробки необхідно провести тестування системи, щоб переконатися у коректності зчитування даних та стабільності передачі їх на мобільний пристрій. У разі виявлення помилок, їх потрібно виправити, уточнивши програмний код та налаштування додатка.

Після успішного тестування метеостанцію встановлюють в потрібному місці для збору реальних даних. Це дозволить перевірити роботу системи в реальних умовах та оцінити її ефективність.

Останнім етапом є документування проекту. Потрібно підготувати детальну документацію, що описує всі етапи проекту, включаючи схеми з'єднання, програмний код та інструкції щодо використання метеостанції. Також потрібно описати результати тестування та надати рекомендації щодо подальшого вдосконалення системи.

Реалізація цього проекту дозволить створити надійну та функціональну метеостанцію, яка буде корисною для широкого кола користувачів.

1 Огляд літератури та постановка задачі

Вимоги до проекту

Технічні та функціональні вимоги

Технічні вимоги:

- Використання платформи Arduino Nano.
- Забезпечення зчитування даних з датчиків температури, вологості та тиску повітря.
- Передача даних на мобільний телефон через USB кабель.

Функціональні вимоги:

- Збір даних про температуру, вологість та тиск повітря в приміщенні.
- Передача отриманих даних на мобільний телефон.
- Розробка інтуїтивного та зручного інтерфейсу для використання на мобільному телефоні.

Додаткові вимоги:

- Ефективне використання ресурсів платформи Arduino Nano.
- Можливість подальшого розширення функціональності метеостанції.
- Надійна та безпечна передача даних через USB кабель.

Очікуваний результат:

Реалізація метеостанції з інтуїтивним інтерфейсом для користувача, яка забезпечує стабільну роботу та передачу даних на мобільний телефон через USB кабель. Метеостанція повинна бути готова до використання в реальних умовах та відповідати всім вимогам проекту.

Математична постановка задачі

Вимірювання даних:

- Нехай T(t), H(t) і P(t) відповідають виміряним значенням температури, вологості та тиску повітря в приміщенні в момент часу t.
- Для кожного часового кроку t, датчики температури T, вологості H і тиску повітря P зчитують значення відповідної величини.

Передача даних на мобільний пристрій:

- Мобільний пристрій функціонує як отримувач даних.
- Дані з метеостанції передаються на мобільний пристрій через USB кабель.
- Розробляється мобільний додаток, який приймає та відображає отримані дані.

Функціональність метеостанції:

- Метеостанція здатна зчитувати значення температури T(t), вологості H(t) і тиску повітря P(t) в реальному часі.
- Зчитані дані формуються у вигляді пакету даних $D(t)=\{T(t),H(t),P(t)\}$.
- Дані передаються на мобільний пристрій через USB кабель для забезпечення користувача актуальними кліматичними данными.

Мета проекту:

- Метеостанція спрямована на надання користувачеві можливості моніторити температуру, вологість та тиск повітря у приміщенні у реальному часі.
- Передача цих даних на мобільний пристрій дозволяє користувачам зручно отримувати та відображати інформацію про кліматичні умови в будь-який момент часу та в будь-якому місці.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1 Зчитування даних з датчиків:

- Програмне забезпечення повинно забезпечувати зчитування значень температури, вологості та тиску повітря з датчиків у реальному часі.

2 Обробка даних:

- Зчитані дані повинні бути оброблені для відображення у зрозумілому форматі.
- Повинна проводитися перевірка на коректність отриманих значень температури, вологості та тиску повітря.

3 Вивід даних на екран мобільного пристрою:

- Програмне забезпечення повинно забезпечувати відображення виміряних значень температури, вологості та тиску повітря на екрані мобільного пристрою.
- Інформація повинна бути представлена у зручному для сприйняття користувача форматі

4 Передача даних на мобільний пристрій:

- Програмне забезпечення повинно забезпечити передачу оброблених даних на мобільний пристрій.
- Забезпечити стабільне та безперервне з'єднання з мобільним пристроєм для передачі даних.

5 Оновлення даних у реальному часі:

- Інформація на екрані мобільного пристрою повинна автоматично оновлюватися при отриманні нових даних з датчиків.
- Забезпечити можливість користувачу оновити дані вручну.

1.2 Розроблення обґрунтування алгоритму функціонування та структурної схеми пристрою

Проект метеостанції на базі Arduino передбачає створення пристрою, який зчитує та обробляє дані про температуру, вологість та атмосферний тиск, а також передає ці дані на мобільний додаток через USB Serial. Для забезпечення стабільної та ефективної роботи системи необхідно розробити обґрунтування алгоритму функціонування та структурну схему пристрою.

Алгоритм функціонування метеостанції

Алгоритм функціонування метеостанції включає кілька основних етапів: зчитування даних з датчиків, обробка цих даних, передача їх на мобільний додаток та відображення даних у реальному часі. Кожен з цих етапів детально розглянемо нижче.

1. **Ініціалізація системи**

При запуску пристрою відбувається ініціалізація всіх компонентів системи, включаючи Arduino Nano, датчики DHT11 та BMP180, модуль ESP8266 та інші необхідні модулі.

Перевірка готовності датчиків до роботи. На цьому етапі мікроконтролер перевіряє, чи всі підключені датчики відповідають та готові до зчитування даних.

2. **Зчитування даних з датчиків**

Датчик DHT11 зчитує значення температури та вологості. Цей датчик періодично вимірює температуру та відносну вологість повітря, передаючи ці дані на мікроконтролер.

Датчик BMP180 зчитує значення атмосферного тиску. Цей датчик також періодично вимірює тиск, передаючи отримані дані на мікроконтролер.

3. **Обробка даних**

Мікроконтролер обробляє зчитані значення, виконуючи необхідні обчислення та конвертації для подальшої передачі.

Перевірка коректності зчитаних даних. На цьому етапі виконується перевірка наявності помилок у зчитаних значеннях, таких як некоректні показники температури чи вологості.

4. **Передача даних на мобільний додаток**

- Налаштування з'єднання через USB serial. Мікроконтролер встановлює зв'язок з комп'ютером або мобільним пристроєм через USB, налаштовуючи його для передачі даних.
- Передача даних через USB на мобільний додаток. Мікроконтролер відправляє оброблені значення температури, вологості та тиску на мобільний додаток через USB з'єднання.

5. **Відображення даних у мобільному додатку**

Мобільний додаток приймає дані з метеостанції та відображає їх у зручному для користувача форматі. Обробка та візуалізація даних у реальному часі. Додаток забезпечує візуалізацію отриманих значень, відображаючи їх у вигляді графіків або текстових повідомлень.

Структурна схема метеостанції

Для реалізації описаного алгоритму функціонування необхідно розробити структурну схему метеостанції, яка включатиме всі основні компоненти та їх з'єднання. На основі вище перелічених функцій, для більшої наочності повного циклу роботи - розробимо блок-схему алгоритму:

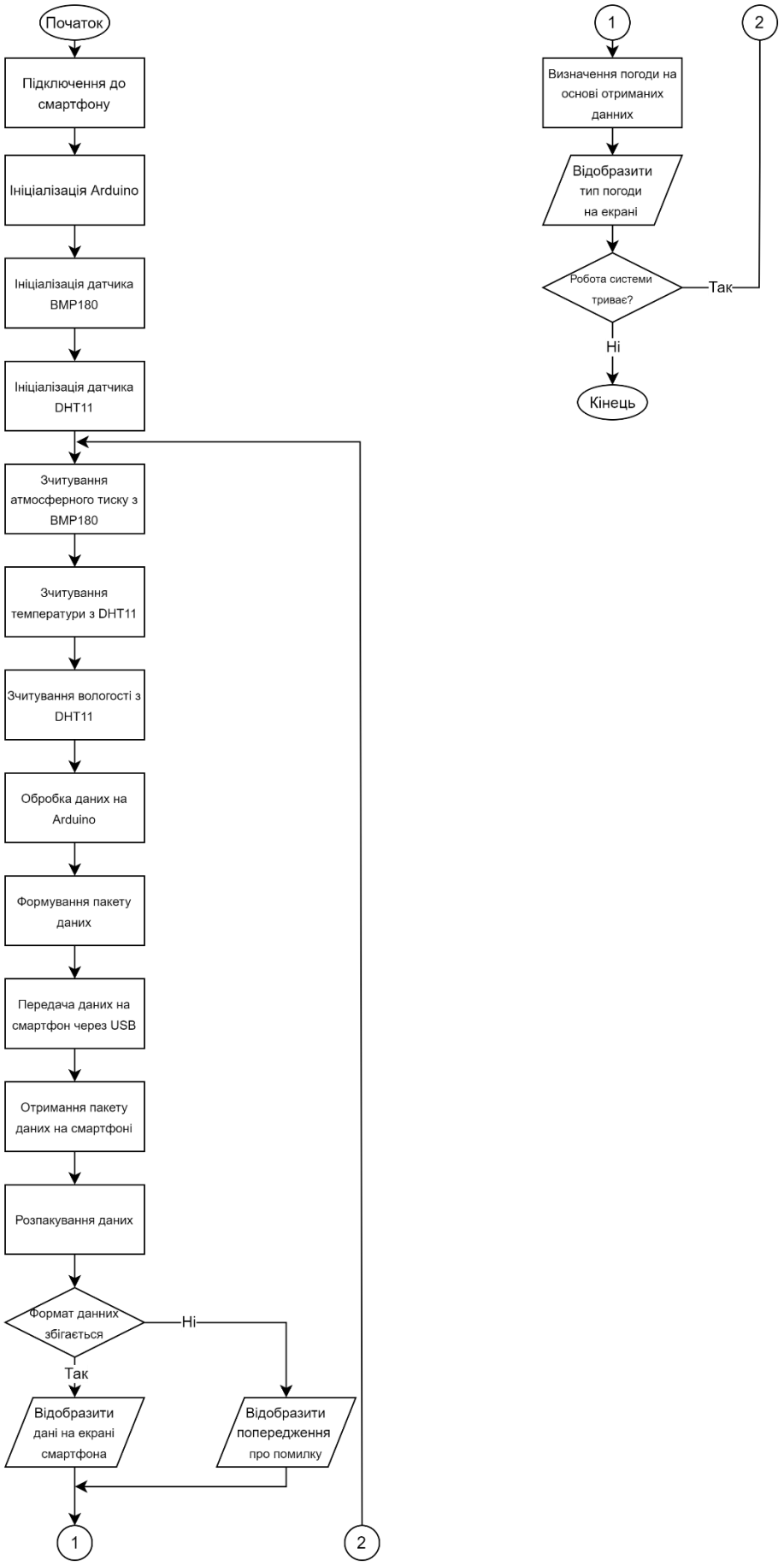

Рисунок 1 – Блок-схема алгоритму системи

Першим пунктом являється підключення системи до смартфону.

Як тільки на вході системи з’являється живлення, вона починає виконувати корисне навантаження.

Другий , третій та четвертий пункт являється перевіркою та індикацією цієї системи

П’ятий пункт, шостий та сьомий пункт – зчитування даних – При завантаженні програмного забезпечення будуть зчитуватись данні з датчиків в Arduino

Восьмий пункт є логічним продовженням. Данні відправляються на Arduino та оброблюються для подальших дій

Дев’ятий пункт, на цьому етапі данні перетворюся в той формат що потрібен та формуються в пакет даних

В десятому пункті пакет даних передається на смартфон а в одинадцятому смартфон отримує в дванадцятому розпаковуватися в додаток та перевіряйтеся чи данні прийшли в правильному форматі, якщо ні то буде помилася, якщо все добре ці данні оброблюється вже додатком де на її основі формується погода та в подальшому вивід н а екран смартфону.

1. **Підключення системи до смартфону**:

Підключення до живлення: Система підключається до джерела живлення. Це може бути батарея, адаптер живлення або будь-яке інше джерело енергії, яке забезпечує достатній рівень напруги для нормальної роботи системи.

Ініціалізація системи: Як тільки система отримує живлення, вона автоматично запускається і починає виконувати свою початкову конфігурацію. На цьому етапі здійснюється перевірка всіх підключених компонентів та модулів.

2. **Встановлення зв'язку зі смартфоном**: Система шукає доступні пристрої, з якими можна встановити з'єднання.
3. **Перевірка наявності живлення**:

Моніторинг напруги: Система перевіряє, чи подається належна напруга на всі важливі компоненти. Для цього можуть використовуватися вбудовані вольтметри або спеціальні датчики напруги.

Сигнал про стан живлення: Якщо живлення подається коректно, система подає сигнал (наприклад, загорається зелений індикатор). Якщо живлення відсутнє або недостатнє, система подає сигнал про помилку (наприклад, блимає червоний індикатор або подає звуковий сигнал).

4. **Перевірка функціональності компонентів**:

Тестування датчиків та модулів: Система послідовно перевіряє кожен підключений датчик та модуль на працездатність. Наприклад, перевіряється відповідь на запит від датчика температури, вологості тощо.

Виявлення несправностей: У випадку, якщо якийсь компонент не відповідає або працює некоректно, система фіксує цю несправність і повідомляє про це користувача через індикатори або повідомлення на екрані.

5. **Індикація готовності**:

Сигнал про готовність: Після успішної перевірки всіх компонентів система подає сигнал про свою готовність до подальшої роботи.

Очікування команд: Система переходить у режим очікування команд від користувача або автоматичних інструкцій для подальшої роботи.

5. **Завантаження програмного забезпечення**:

Ініціалізація програмного забезпечення: Після запуску системи завантажується основне програмне забезпечення, яке відповідає за зчитування та обробку даних. Це може бути прошивка для Arduino, яка управляє всіма підключеними датчиками.

Підготовка до зчитування: Програмне забезпечення перевіряє готовність датчиків до роботи, встановлює необхідні параметри для коректного зчитування даних.

6. **Зчитування даних з датчиків**:

Запит на зчитування: Програмне забезпечення посилає запити до кожного датчика для отримання необхідних даних. Наприклад, запит на вимірювання температури або вологості.

Отримання даних: Датчики відповідають на запити та передають зчитані значення на Arduino. Ці значення можуть бути в різних форматах, залежно від типу датчика.

7. **Обробка зчитаних даних**:

Попередня обробка: Отримані від датчиків дані передаються на Arduino для попередньої обробки. Наприклад, можуть проводитися перетворення аналогових сигналів у цифрові, фільтрація шумів або нормалізація даних.

Збереження даних: Оброблені дані зберігаються у тимчасовій пам'яті для подальшої обробки та передачі.

8. **Обробка даних на Arduino**:

Форматування даних: Arduino обробляє отримані дані, приводячи їх до єдиного формату. Наприклад, об'єднує дані з різних датчиків у структури або масиви.

Підготовка до передачі: Дані готуються до передачі на смартфон, можливо з додатковим шифруванням або стисненням для зменшення обсягу переданих даних.

9. **Формування пакету даних**:

Створення пакету: Дані перетворюються у потрібний формат і формуються у пакет даних. Пакет може містити інформацію про температуру, вологість, тиск та інші показники.

Перевірка цілісності: Пакет даних перевіряється на цілісність та коректність, додаються контрольні суми або інші механізми для виявлення помилок при передачі.

10. **Передача пакету даних на смартфон**:

Встановлення з'єднання: Arduino встановлює з'єднання зі смартфоном

Відправлення даних: Пакет даних передається на смартфон, використовуючи встановлений канал зв'язку.

11. **Отримання пакету даних смартфоном**:

Прийом даних: Смартфон отримує пакет даних від Arduino. Це може бути зроблено через спеціальний додаток, який контролює процес прийому даних.

Перевірка цілісності: Смартфон перевіряє цілісність та коректність отриманих даних, використовуючи контрольні суми або інші механізми виявлення помилок.

12. **Розпаковка та обробка даних у додатку**:

Розпаковка пакету: Додаток на смартфоні розпаковує отриманий пакет даних, розділяючи його на окремі значення (температура, вологість тощо).

Перевірка даних: Додаток перевіряє, чи всі дані отримані в правильному форматі. Якщо виявлено помилки, відображається відповідне повідомлення.

Обробка даних: Якщо дані коректні, додаток обробляє їх для подальшого використання. Наприклад, створює графіки, обчислює середні значення або виводить прогнози.

Вивід даних на екран: Оброблені дані виводяться на екран смартфону у вигляді зручного інтерфейсу, наприклад, у формі діаграм, графіків або текстових повідомлень про поточну погоду.

2 Розробка структурної схеми пристрою

Відповідно до технічного завдання, необхідно розробити систему, яка задовольняє початкові потреби середньостатистичної людини з інвалідністю. Система повинна бути, в першу чергу, зручною, швидкою та доступною.

безпровідний режим роботи;

- високий рівень безпеки доступу до керування;
- простою у використанні;
- мати низьке споживання електроенергії;
- зручну масштабованість;
- компактність.

SHS повинна забезпечувати виконання наступних функцій:

- під’єднання до смартфону;
- прийому команд від користувача за допомогою додатку;
- Відображення інформації про кліматичні умови

На основі вищевикладеного, складемо структурну схему

Система не є доволі складною, з точки зору затрачених ресурсів та обчислювальної потужності на протязі часу, адже значна частина програмного коду буде знаходитись на смартфоні користувача але в той же час необхідно забезпечити:

- живлення системи від одного джерела 5 В;
- узгодження роботи смартфону та мікроконтролера.

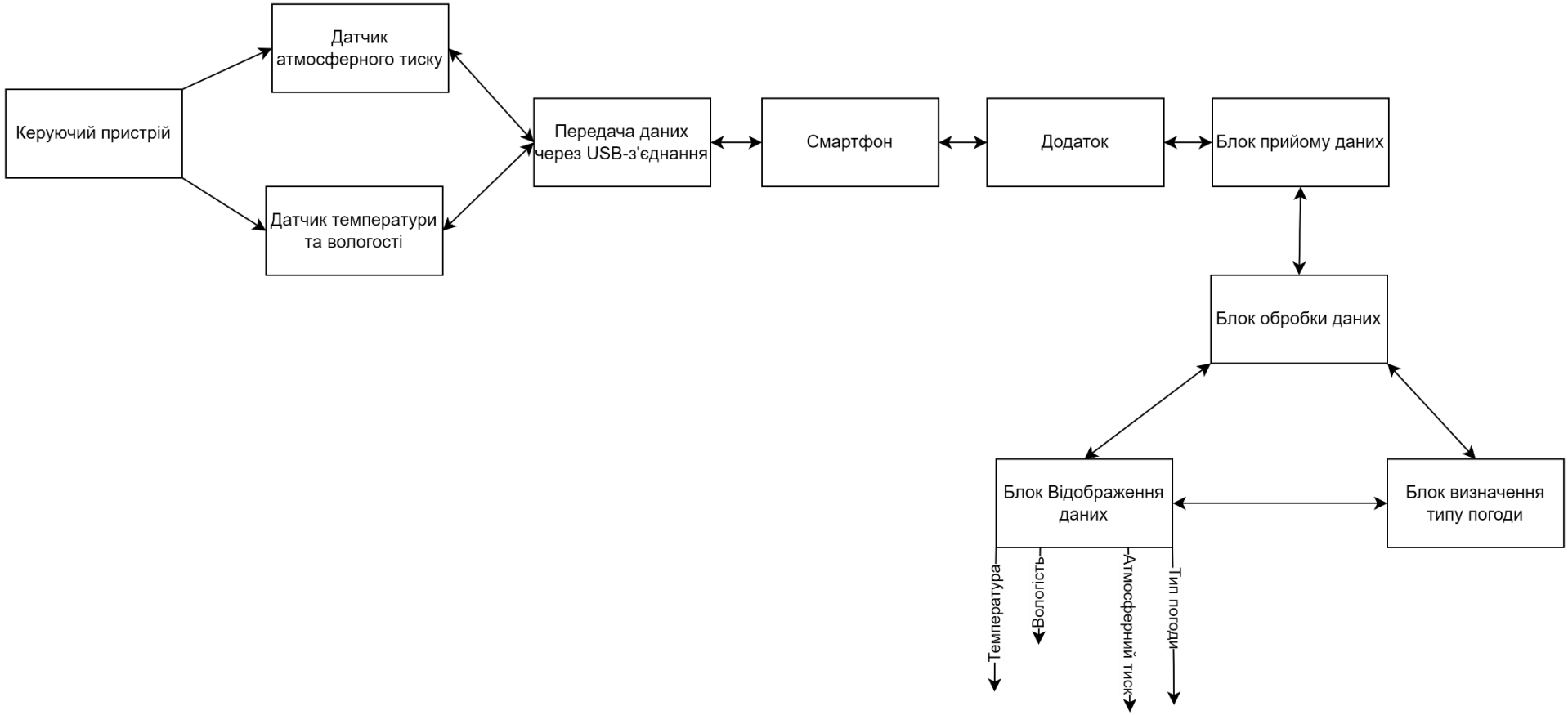

Рисунок 2 – Структурна схема

2.1 Розроблення схеми електричної принципової пристрою що проектуються та вибір елементної бази

У процесі розробки метеостанції на базі Arduino Nano важливо правильно вибрати елементну базу, яка забезпечить надійність і точність вимірювань. Ключовими компонентами системи є контролер, датчики та засоби зв'язку. Розглянемо вибір кожного з цих елементів. Вона має вбудований USB-порт для зручного програмування і підключення до комп'ютера. Arduino Nano обрано як головний мікроконтролер для проекту, оскільки вона має достатньо ресурсів для обробки даних з датчиків.

1. Платформа Arduino Nano:

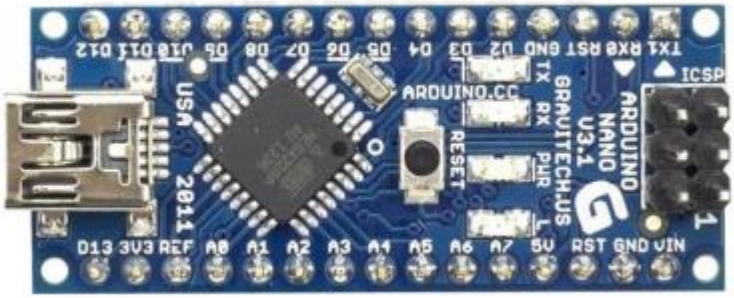

Рисунок 3 – Arduino Nano:

- Arduino Nano - це мікроконтролерний модуль, який базується на ATmega328. Він має компактний розмір і вбудований USB-інтерфейс для зручного програмування. Arduino Nano використовується як головний контролер у проекті метеостанції. Він приймає дані від датчиків та керує процесом передачі цих даних на мобільний пристрій. Незважаючи на свій скромний розмір, вона практично нічим не поступається Arduino Uno за функціоналом і може використовуватися в проектах, де габарити відіграють істотну роль

Переваги:

- Невеликі розміри, що дозволяють створювати компактні пристрої.
- Велика кількість цифрових і аналогових пінів для підключення датчиків та інших компонентів.
- Вбудований USB-конектор для програмування та передачі даних.

- Підтримка великої кількості бібліотек та прикладів, що спрощує розробку.

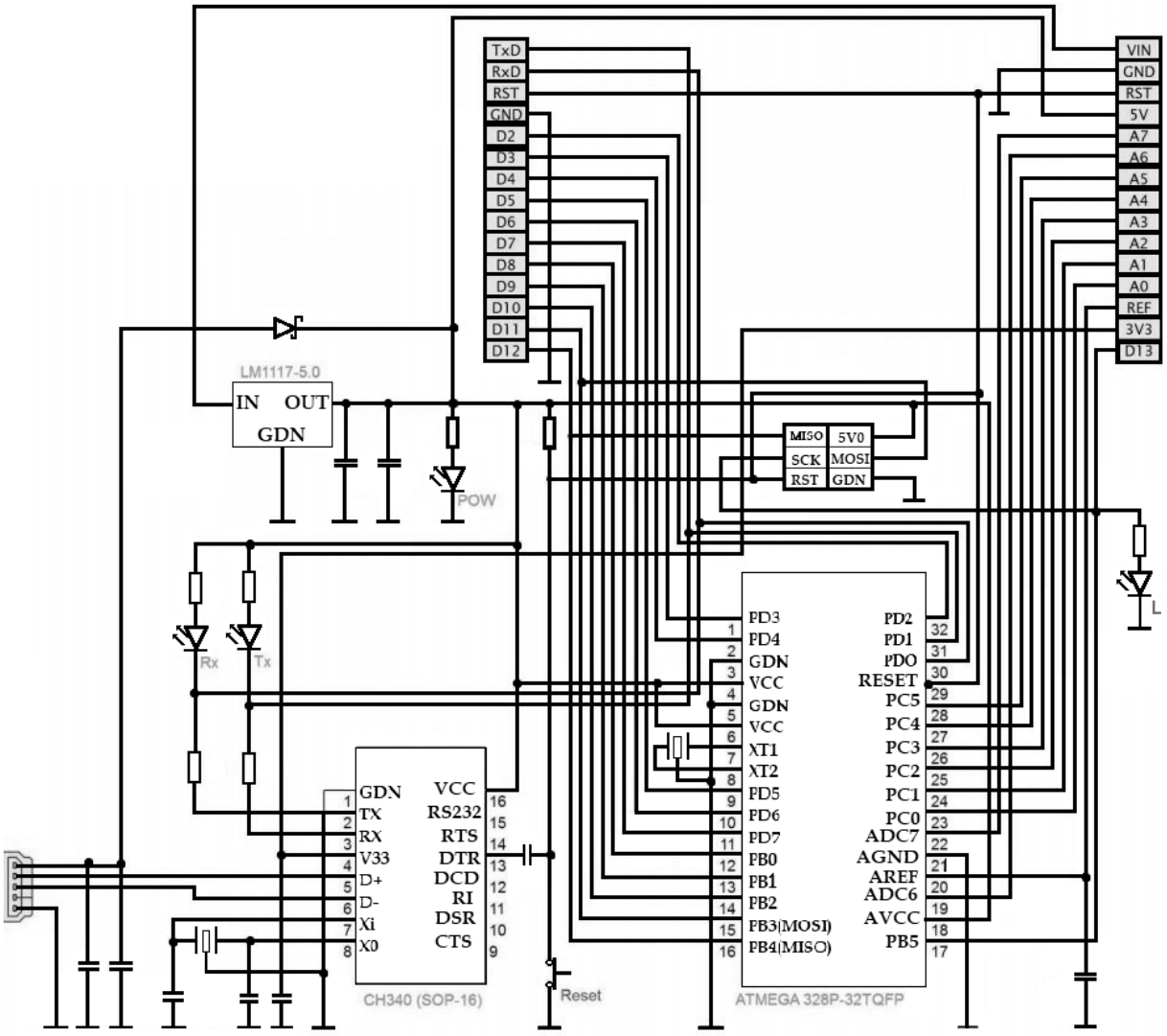

Рисунок 4 – Принципова схема Arduino Nano

Платформа Nano, побудована на мікроконтролері ATmega328, має невеликі розміри, що дозволяє використовувати її в компактних пристроях.

Основні характеристик Arduino Nano

Категорія	Параметр
Мікроконтролер	ATmega328P,8-бітний AVR на частоті 16 МГц
Пам’ять	Flash – 30 кбайт
	SRAM – 2 кбайт
	EEPROM – 1 кбайт
Порти	Портів введення-виводу всього 20
	З АЦП – 8
	З ШІМ – 6, 8-біт
Інтерфейси	SPI
	I²C / TWI
	UART / Serial
Потужність	Номінальна робоча напруга – 5 В Допустима напруга – 7-12В
	Максимальний струм – 800 мА

2. Датчик DHT11 (Digital Humidity Temperature Sensor):
   DHT11 - це цифровий датчик, призначений для виміру температури та вологості повітря. Він має простий протокол зв'язку і досить низьку вартість. DHT11 з'єднується з Arduino Nano за допомогою цифрових входів / виходів і передає виміряні дані про температуру та вологість на контролер для подальшої обробки та передачі. Датчик цієї моделі має температурну компенсацію та калібрування в точній калібрувальній камері, а коефіцієнт калібрування зберігається в пам’яті.

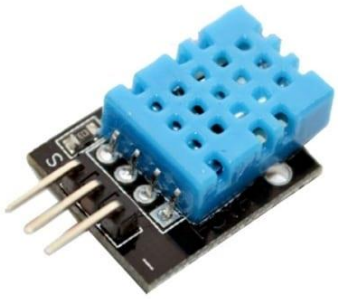

Рисунок 5 – Датчик температури та вологості DH11

Для контакту з платою використовуються виводи: - VCC – для живлення датчика від 3,3 В; 46 - Data Out – для виводу даних; - GND – земля.

Переваги:

Легкість у використанні та інтеграції з Arduino.

Доступна ціна.

Достатня точність для побутових застосувань (±2°C для температури та ±5% для вологості).

Обмеження:

Відносно низька точність у порівнянні з іншими моделями.

Обмежений діапазон вимірювань (температура від 0 до 50°C, вологість від 20 до 80%).

Основні характеристик DHT11

Модель датчика:	DHT11
Діапазон вимірювання вологості/ точність	5 - 95% RH ± 5% (макс.)
Діапазон вимірювання температури/точність	0 ~ +60 ºC ± 2% (макс.)
Напруга живлення	3.5-5.5 В
Частота опитування сенсора	не більше 1 Гц
Розміри корпуса	15.5 x 12 x 5.5 мм

3. Датчик BMP180 (Barometric Pressure Sensor):
   BMP180 - це високоточний датчик атмосферного тиску і температури. Він здатен вимірювати тиск повітря в діапазоні від 300 до 1100 гектопаскалей і температуру від -40°C до +85°C. BMP180 підключається до Arduino Nano через I2C інтерфейс і передає дані про тиск і температуру для подальшої обробки.

Рисунок 6 – Датчик атмосферного тиску BMP180

Переваги:

Висока точність вимірювання атмосферного тиску (±1 hPa).

Можливість вимірювання висоти над рівнем моря.

Компактний розмір і легкість інтеграції.

Обмеження:

Вимірює лише тиск і температуру

Основні характеристик BMP180

Модель датчика:	GY-65
Діапазон вимірювання/ точність	300 - 1100 гПа (9000 - 500 метрів над рівнем моря)
Пропускна здатність	0.03 гПа / 0.25 м
Діапазон виміру температур	-40 to + 85 ° C (точність ± 2 град)
Напруга живлення	2 - 5В
Споживання в режимі очікування	0,1 мкА

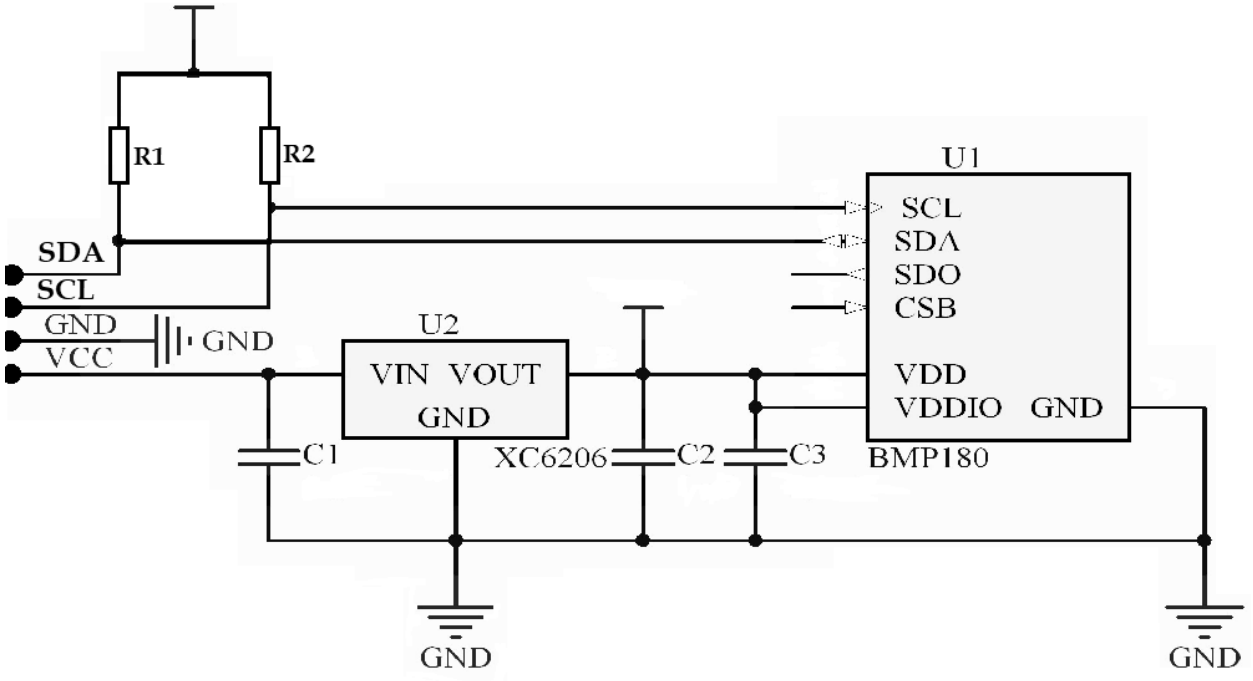

Рисунок 7 – Принципова схема BMP180

У проекті використовується процесорна одиниця в мікроконтролері Arduino Nano

Arduino Nano (ATmega328P):

- Arduino Nano базується на мікроконтролері ATmega328P від Microchip (раніше від Atmel). Цей процесор має тактову частоту 16 МГц та 32 кілобайти вбудованої флеш-пам'яті для програмного забезпечення. ATmega328P є досить потужним мікроконтролером, який може виконувати широкий спектр завдань, включаючи зчитування даних з датчиків, обробку і передачу даних.

Ця процесорна одиниця забезпечують обробку даних з датчиків та управління відповідними пристроями для надійного та ефективного функціонування метеостанції.

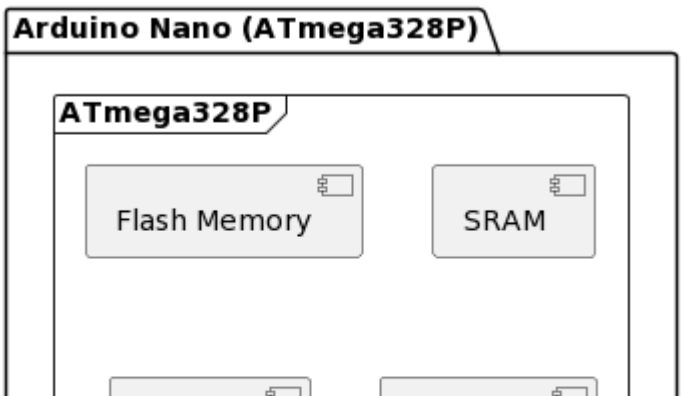

Рисунок 8 – Схема зображенням ATmega328P

Вибір елементної бази для метеостанції на базі Arduino Nano визначається необхідністю забезпечити точність, надійність та зручність у використанні. Arduino Nano є оптимальним вибором контролера завдяки своїй компактності та функціональності. Датчики DHT11 та BMP180 забезпечують необхідний набір параметрів для моніторингу метеорологічних умов. USB-зв'язок дозволяє легко передавати дані на мобільний пристрій. Додаткові компоненти, такі як резистори та провідники, забезпечують стабільну роботу системи.

Проект метеостанції вимагає створення функціональної схеми, що об'єднує різні датчики та модулі для збору та передачі метеорологічних даних на мобільний пристрій. Основними компонентами системи є Arduino Nano, датчики DHT11 та BMP180, а також засіб зв'язку з телефоном через USB.

Опис компонентів

1. **Arduino Nano**: Це основний контролер системи, який здійснює збір та обробку даних з датчиків.
2. **DHT11**: Датчик температури та вологості, який забезпечує вимірювання цих параметрів з достатньою точністю для побутового використання.
3. **BMP180**: Барометричний датчик тиску, який також може вимірювати температуру і висоту.
4. **USB-підключення**: Використовується для передачі даних з Arduino на телефон.

Схема підключення

Схема електричної функціональної пристрою включає в себе наступні елементи та їх підключення:

1. **Підключення DHT11 до Arduino Nano**:

    - VCC (живлення) — підключено до 5V на Arduino.
    - GND (заземлення) — підключено до GND на Arduino.
    - DATA (сигнальний пін) — підключено до цифрового піна D2 на Arduino через резистор 10 кОм (pull-up resistor).

2. **Підключення BMP180 до Arduino Nano**:

    - VCC (живлення) — підключено до 3.3V на Arduino.
    - GND (заземлення) — підключено до GND на Arduino.
    - SCL (Serial Clock Line) — підключено до аналогового піна A5 на Arduino.

    - SDA (Serial Data Line) — підключено до аналогового піна A4 на Arduino.

3. **USB-підключення**:

    - Використовується стандартне USB-підключення, яке забезпечує як живлення, так і передачу даних між Arduino Nano та мобільним пристроєм.

Опис роботи

Пристрій функціонує таким чином:

1. Arduino Nano зчитує дані з датчиків DHT11 та BMP180 через відповідні пінові інтерфейси.
2. Зібрані дані обробляються та конвертуються у відповідні формати.

Через USB-підключення дані передаються на мобільний телефон, де вони можуть бути відображені за допомогою спеціального додатку або через серійний монітор. Така схема дозволяє ефективно збирати метеорологічні дані та передавати їх на мобільний телефон для подальшого аналізу і відображення.

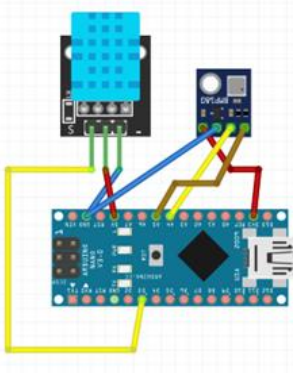

Рисунок 9 – Схема зображенням метеостанції

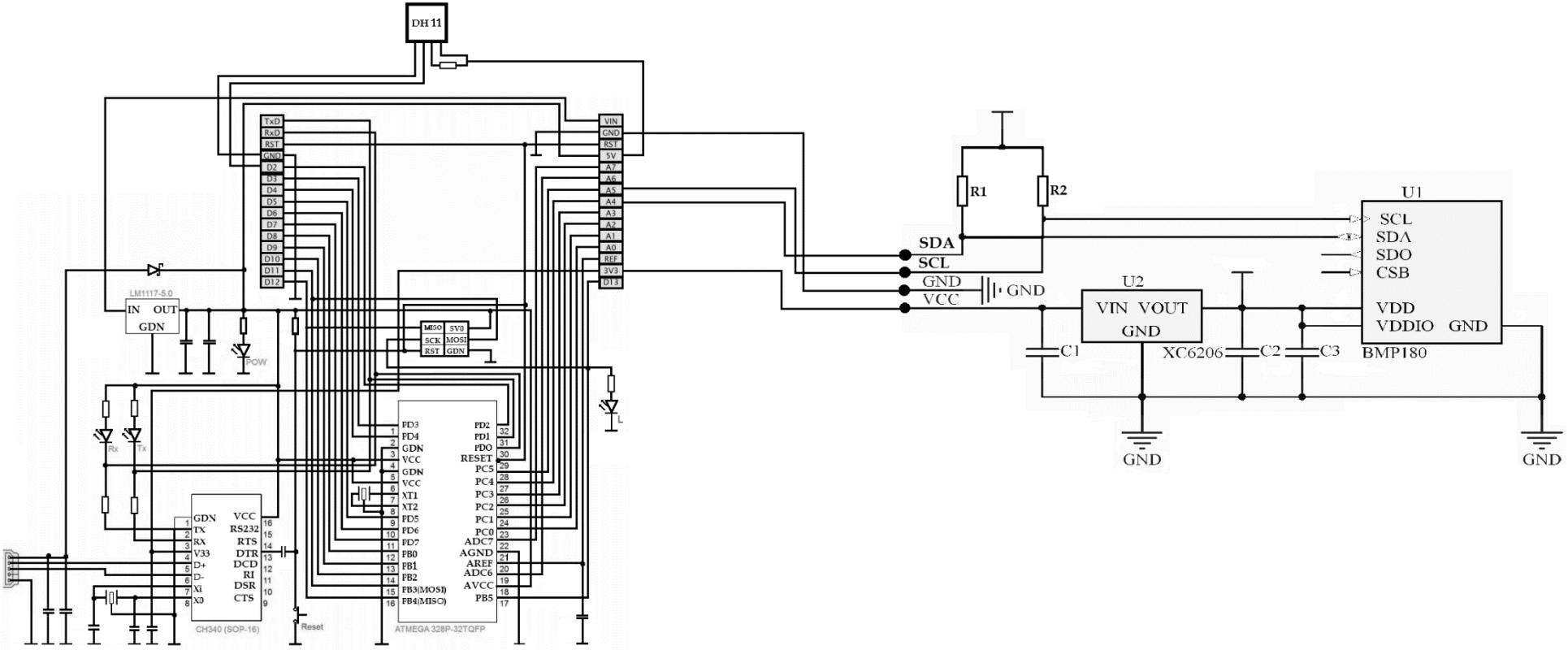

Рисунок 10 - Принципова схема

2.3 Розрахунки та синтез основних електронних вузлів, блоків, схем керування.

Для успішної реалізації метеостанції на базі Arduino Nano необхідно провести розрахунки і синтез основних електронних вузлів, включаючи блоки живлення, схеми керування, синхронізації та інші. Це забезпечить коректну роботу всіх компонентів системи. Цей поект включає кілька ключових етапів, пов'язаних з розрахунками та синтезом електронних компонентів. У цьому розділі розглянемо основні електронні вузли, блоки, схеми керування та синхронізації, які забезпечують коректну роботу пристрою.

Розрахунки живлення

1. **Живлення Arduino Nano**:

Напруга живлення: 5V через USB або 7-12V через VIN.

Споживана потужність: Максимальна струмова споживаність Arduino Nano становить приблизно 19 мА.

2. **Живлення DHT11**:

Напруга живлення: 3.3V або 5V.

Споживана потужність: До 2.5 мА при вимірюванні.

3. **Живлення BMP180**:

Напруга живлення: 1.8V - 3.6V (3.3V зазвичай).

Споживана потужність: 12 μA в режимі вимірювання.

Синтез схеми керування

Основна схема керування базується на програмному забезпеченні, що працює на Arduino Nano. Програма повинна забезпечити:

Читання даних з датчиків:Використання бібліотек для роботи з DHT11 та BMP180. Ініціалізація датчиків та періодичне зчитування даних.

Обробку даних: Конвертація сирих даних у зрозумілі користувачеві значення (температура, вологість, тиск).Виявлення аномалій та забезпечення фільтрації шуму.

Передачу даних:Виведення даних на серійний порт для передачі на мобільний пристрій. Використання USB-з'єднання для стабільної передачі даних.

Підключення Arduino Nano до смартфону через USB: Використовується стандартний USB-кабель для підключення Arduino Nano до смартфону з ОС Android. На стороні смартфону використовується Android-додаток для зчитування та відображення даних з серійного порту

1. Енергоспоживання

Розрахунок енергоспоживання:

- Arduino Nano: ~19 мА
- DHT11: ~0.5 мА (під час вимірювання)
- BMP180: ~12 мкА (в режимі очікування), ~5 мА (під час вимірювання)
- Загальне енергоспоживання залежить від частоти зчитування даних та режиму роботи датчиків.

Схема синхронізації

Синхронізація роботи датчиків та обробки даних досягається через програмний таймер, що забезпечує

1. **Періодичність вимірювань**:

   Встановлення інтервалів зчитування даних (наприклад, кожні 2 секунди для DHT11 та кожну секунду для BMP180).

   Забезпечення коректного порядку зчитування та обробки даних.

2. **Управління енергоспоживанням**:

   Вимкнення невикористовуваних модулів або перехід в енергозберігаючий режим між вимірюваннями для зменшення споживання енергії.

Розрахунки та синтез основних електронних вузлів, блоків, схем керування та синхронізації показали, що метеостанція на базі Arduino Nano є ефективним та надійним пристроєм для моніторингу кліматичних умов. Використання доступних компонентів та простих схем підключення забезпечує стабільну роботу системи та точне зчитування даних. Передача даних через USB-з'єднання дозволяє легко інтегрувати пристрій з мобільними додатками, що значно розширює його функціональні можливості.

3 Розроблення програмного забезпечення

3.1 Проектування інтерфейсу

Розробка інтерфейсу користувача здійснюється в середовищі розробки Android Studio з урахуванням забезпечення зручного та ефективного взаємодії з програмним забезпеченням. Інтерфейс повинен бути зрозумілим та ергономічним, з урахуванням кращих практик дизайну інтерфейсів для мобільних додатків. Навігація між різними екранами та функціями повинна бути логічною та інтуїтивно зрозумілою для користувача. Використання елементів дизайну Material Design або власних кастомних компонентів для забезпечення модерного та привабливого вигляду інтерфейсу.

Основні елементи інтерфейсу

- TextView для відображення температури, вологості, тиску повітря та іншої інформації.
- Button для оновлення даних.

Кольори та шрифти

- Колір тексту: purple_700 (фіолетовий) для текстових елементів.
- Колір фону: white (білий) для фону інтерфейсу.
- Шрифт: Розмір шрифту 18sp для заголовків та 24sp для основних значень.

Розміри та розташування елементів

- Розташування текстових елементів в центрі екрану за допомогою параметра android:layout_centerHorizontal="true".
- Відступи між елементами та від країв екрану: 16dp.
- Кнопка оновлення розміщена під текстом про погоду з центрованим розташуванням.

Загальні принципи проектування інтерфейсу

При проектуванні інтерфейсу було дотримано такі принципи:

Інтуїтивність: інтерфейс простий і зрозумілий для користувача. Користувачеві не потрібно читати довгі інструкції, щоб зрозуміти, як грати.

Привабливість: інтерфейс приємний для очей. Гра виконана в яскравих кольорах, які створюють позитивний настрій.

Послідовність: інтерфейс послідовний у використанні елементів керування. Наприклад, для керування Пакманом використовуються клавіші зі стрілками, які є стандартом для ігор.

3.2 Проектування структур системи

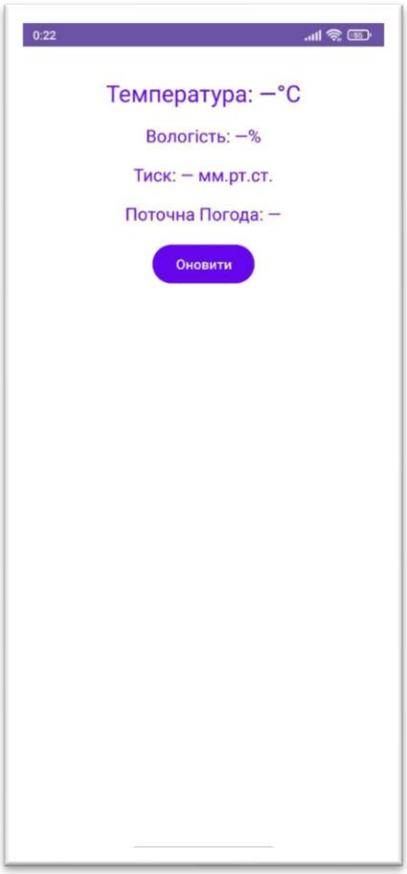

Рисунок 12 - Інтерфейс Додатка

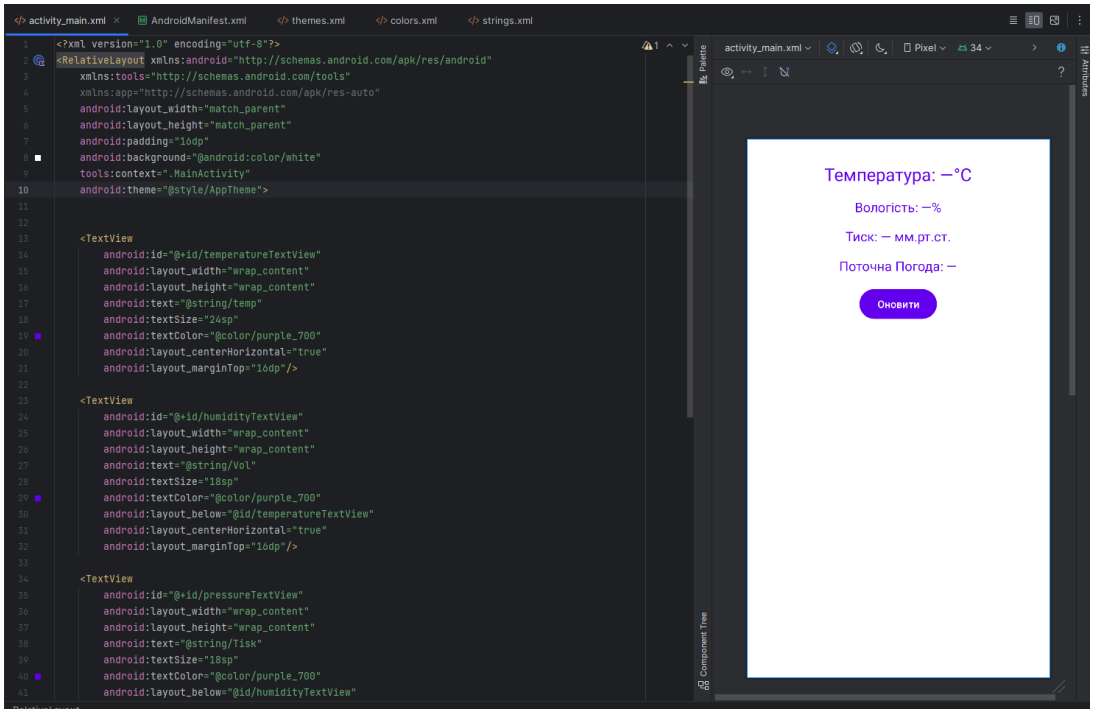

Рисунок 13 – Проектування інтерфейсу додатка

На вказаному вікні міститься інтерфейс мобільного додатку для метеостанції. Основні елементи на цьому вікні включають:

TextView для відображення температури, вологості, тиску повітря та іншої інформації: У даному випадку, на екрані відображаються чотири TextView, кожна з яких відповідає певному параметру: температурі, вологості, тиску повітря та погоднім умовам. Ці тексти розміщені один під одним і центруються горизонтально в центрі екрану. Кожен текст має відповідний розмір шрифту та колір, які визначені в ресурсах додатка.

Button для оновлення даних: Кнопка оновлення даних розміщена під останнім текстом про погоду. Вона дозволяє користувачеві оновити інформацію про погоду за потреби. Кнопка має текст "Оновити" і стилізована з використанням фонового кольору та кольору тексту, що визначені в ресурсах додатка.

Ці елементи складають простий та зрозумілий інтерфейс, який дозволяє користувачеві легко переглядати інформацію про погоду та оновлювати її за потреби.

Проектування алгоритмів додатка

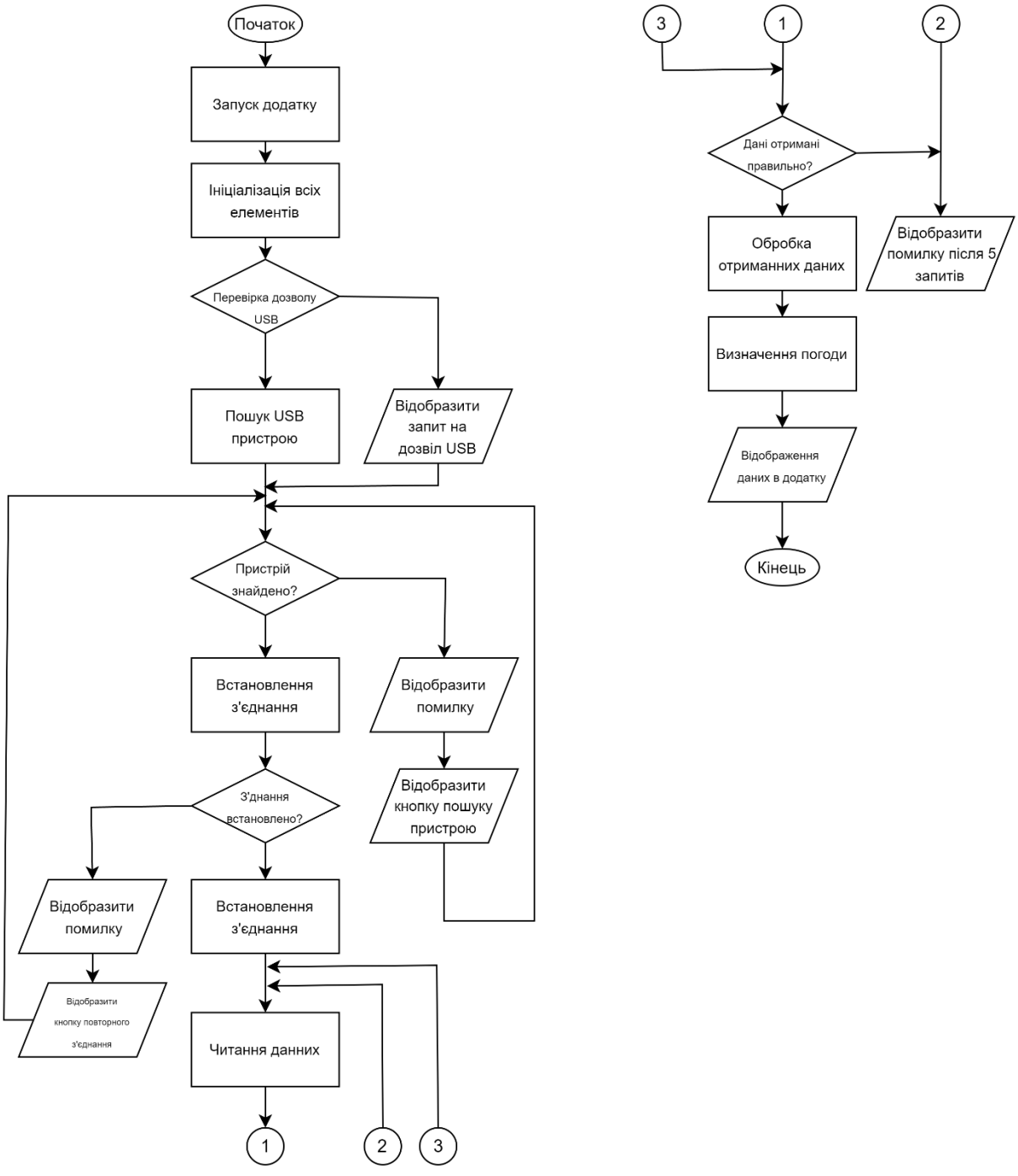

Рисунок 14 – Блок схема логіки додатка

Цей додаток є потужним інструментом для спілкування між Android-пристроями та пристроями Arduino через USB-з'єднання. Він створений для того, щоб спростити зчитування даних з Arduino і відображення їх на екрані Android-пристрою.

В основі додатку лежить зручний інтерфейс, який надає користувачеві можливість переглядати важливі дані про навколишнє середовище, такі як температура, вологість та тиск, отримані від датчиків, підключених до Arduino. Ці дані оновлюються в реальному часі, щоб користувач завжди мав актуальну інформацію.

Однією з ключових особливостей є сповіщення, які додаток використовує для повідомлення користувача про будь-які зміни стану підключення пристрою Arduino. Це дозволяє забезпечити зручність інтерфейсу та швидку реакцію на будь-які події, пов'язані з пристроєм.

Крім того, додаток має можливість взаємодії з користувачем через кнопки "Оновити" та "Вибрати пристрій". Кнопка "Оновити" дозволяє користувачеві вручну оновити дані, а кнопка "Вибрати пристрій" дає можливість вибрати підключений пристрій Arduino, якщо на пристрої є кілька пристроїв, доступних для підключення.

Загалом, цей додаток відкриває нові можливості для взаємодії з пристроями Arduino через Android-пристрої. Його зручний інтерфейс, сповіщення та функціональність роблять його ідеальним інструментом для будь-якого, хто цікавиться взаємодією з електронікою за допомогою мобільного пристрою.

ndroid-фреймворк для взаємодії з USB-пристроями та роботи з дозволами на доступ до них. Ось як він працює крок за кроком:

1. **Пошук пристрою Arduino**: При запуску додаток сканує всі підключені USB-пристрої. Якщо знаходиться пристрій з відповідним вендорським ID (в цьому випадку, ID для Arduino), то додаток встановлює з'єднання з ним.
2. **Надання дозволу на підключення**: Якщо пристрій Arduino виявлено, але додаток ще не має дозволу на його використання, то він запитує користувача про надання цього дозволу.
3. **Відкриття з'єднання та зчитування даних**: Після отримання дозволу, додаток відкриває з'єднання з Arduino та починає зчитування даних через USB. Він використовує окремий потік для зчитування, щоб не блокувати головний інтерфейс.
4. **Оновлення інтерфейсу користувача**: Отримані дані (температура, вологість, тиск тощо) оновлюються на екрані додатку у реальному часі, щоб користувач міг бачити актуальну інформацію.
5. **Сповіщення про зміни стану підключення**: Якщо стан підключення пристрою змінюється (приєднання або від'єднання Arduino), додаток надсилає сповіщення користувачеві, щоб повідомити про цю подію.
6. **Взаємодія з користувачем**: Крім автоматичного оновлення даних, користувач може вручну оновити їх, натиснувши кнопку "Оновити", а також вибрати підключений пристрій через кнопку "Вибрати пристрій".

Проектування алгоритмів метеостанції

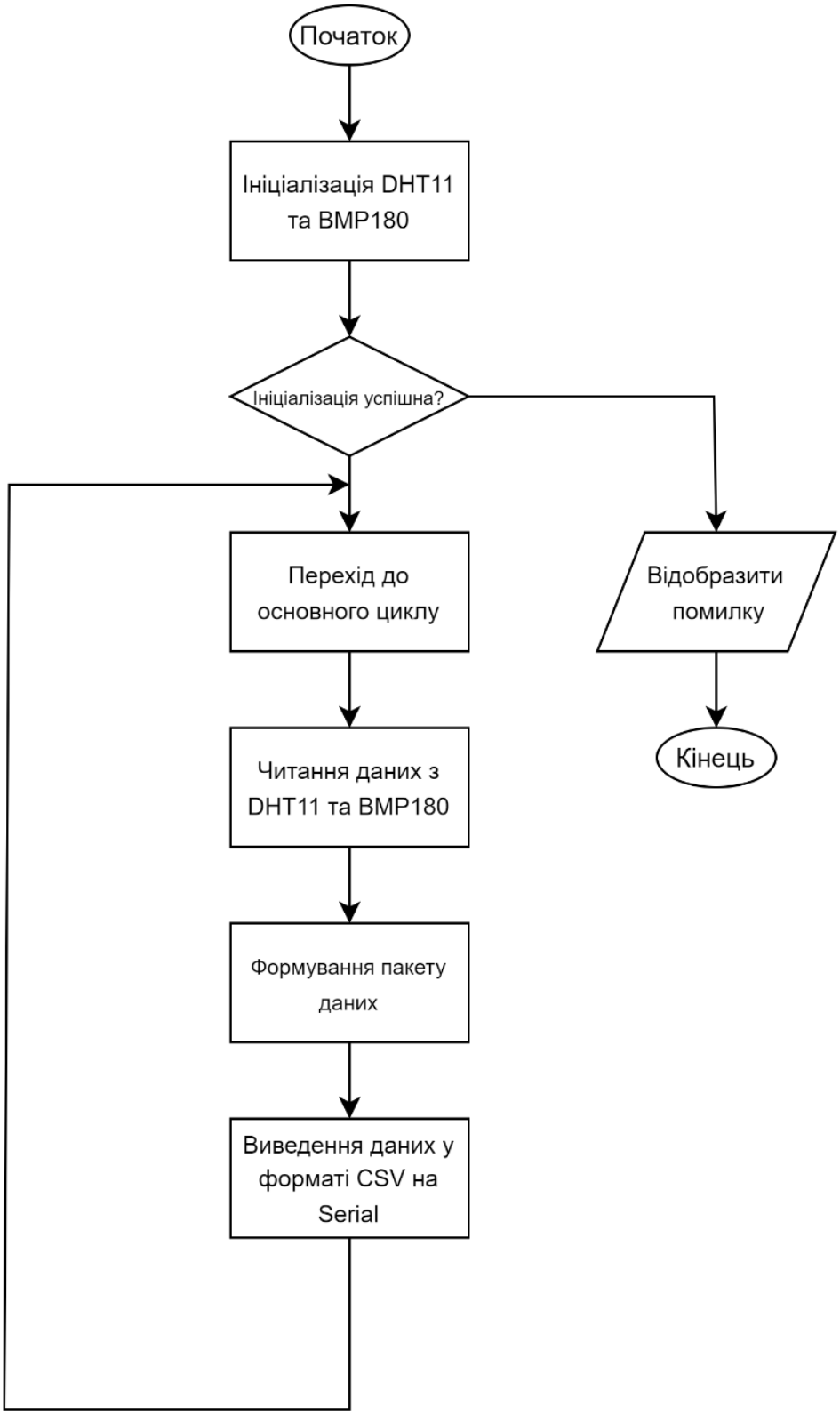

Рисунок 15 – Блок схема логіки метеостанції

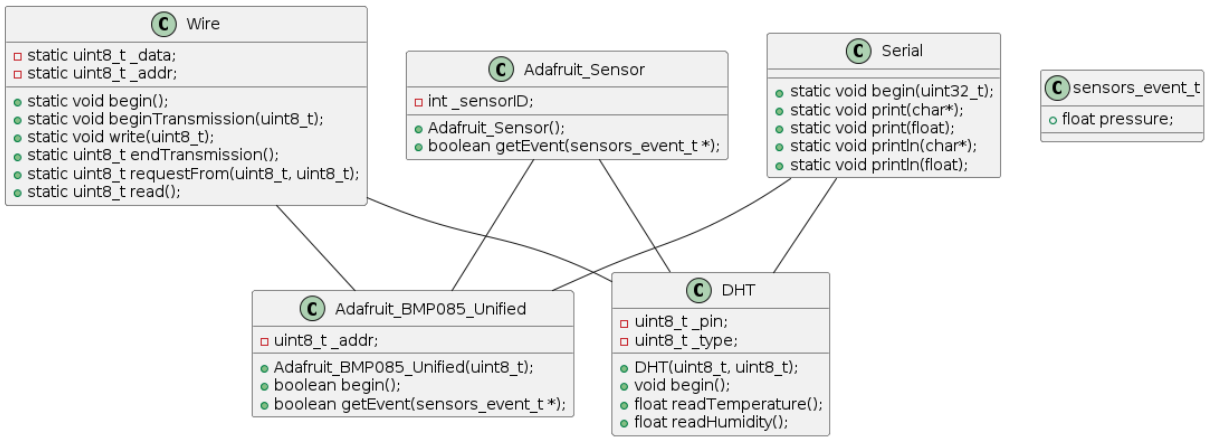

Рисунок 16 – Діаграма класів метеостанції

Клас DHT: Цей клас відповідає за управління датчиком DHT11 для вимірювання температури та вологості повітря. У нього є два приватних поля: pin (цифровий пін, до якого підключений датчик) та type (тип датчика, у цьому випадку, DHT11). Клас має конструктор, який приймає значення pin та type. Також він має методи для ініціалізації датчика (begin()), зчитування вологості (readHumidity()) та температури (readTemperature()) з датчика.

Клас Adafruit_BMP085_Unified: Цей клас відповідає за управління датчиком BMP180 для вимірювання атмосферного тиску. У нього є приватне поле sensorID (ідентифікатор датчика). Клас також має конструктор, який приймає sensorID, та методи для ініціалізації датчика (begin()) та отримання даних з нього (getEvent()).

Клас sensors_event_t: Цей клас представляє події датчика, зокрема, вимірювання тиску. У нього є публічне поле pressure (тиск) та конструктор за

замовчуванням.

Ці класи спільно створюють програмну структуру для взаємодії з датчиками та обробки вимірювань температури, вологості та тиску в повітрі.

Процес роботи програми полягає в наступному:

1 При запуску виконується метод setup(), в якому налаштовуються датчики

2 Після завершення методу setup() виконується метод loop(), який запускає безкінечний цикл для отримання даних з датчиків та надсилання їх на в додаток
3 У циклі отримуються значення температури, вологості та тиску за допомогою методів датчиків DHT та BMP180.
4 Процес повторюється в безкінечному циклі, забезпечуючи постійне оновлення даних на сервері.

Принцип роботи

Ініціалізація: При запуску метеостанції на Arduino (скетча) викликається метод setup(). У цьому методі налаштовуються всі необхідні компоненти, такі як датчики температури, вологості та тиску. Датчики підключаються до відповідних портів мікроконтролера, ініціалізуються, і, можливо, вони калібруються для отримання точних вимірювань.

Збір даних: Після налаштування компонентів мікроконтролер починає безперервно зчитувати дані з датчиків. Ці дані можуть бути значення температури, вологості та тиску, які отримуються з відповідних датчиків. Дані можуть бути зчитані з певною періодичністю, наприклад, кожні 5 секунд.

Надсилання даних: Після отримання даних від датчиків, ардуино надсилає їх через серіний порт до додатку

Обробка та відображення даних: В мобільному пристрої отримані дані можуть бути оброблені та відображені у зручному для користувача форматі.

Цикл повторення: Після відправки даних метеостанціям може бути викликаний метод loop(), в якому виконується безкінечний цикл, що забезпечує постійне зчитування даних з датчиків та їх відправку на віддалений сервер або мобільний пристрій. Цей процес продовжується, доки метеостанція не буде вимкнена.

Рисунок 7 – Діаграма роботи проекту

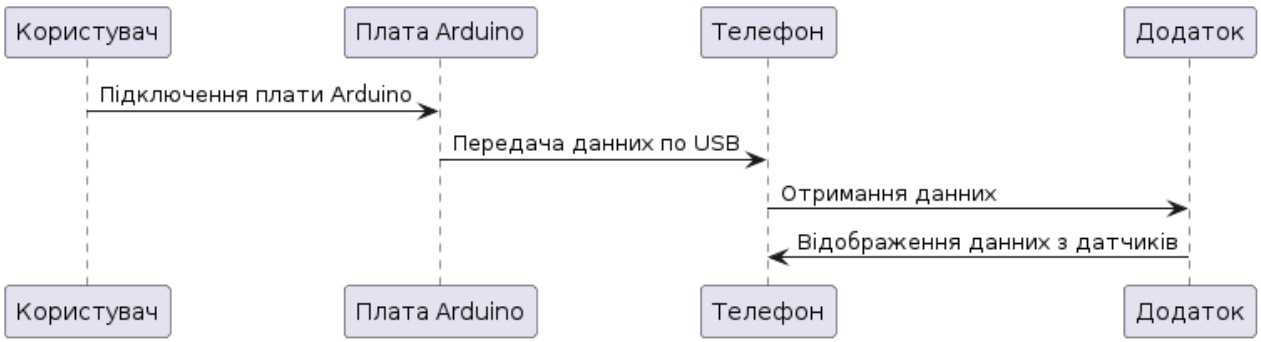

Рисунок 17 – Діаграма роботи проекту

3.1 Вибір та розроблення алгоритмів

У процесі вибору та розроблення алгоритмів для мікропроцесорного пристрою метеостанції було проведено ретельний аналіз доступних методів зчитування даних з датчиків температури, вологості та тиску. Були враховані різні фактори, такі як точність вимірювань, швидкість реакції, а також складність інтеграції з мікроконтролером.

Для датчиків температури та вологості було обрано алгоритми, що базуються на протоколі зчитування DHT. Ці алгоритми дозволяють ефективно взаємодіяти з датчиками DHT і отримувати стабільні та точні вимірювання температури та вологості навколишнього середовища.

Для зчитування даних тиску було обрано алгоритми, що базуються на протоколі зчитування BMP180. Ці алгоритми дозволяють ефективно взаємодіяти з датчиками BMP180 і отримувати стабільні та точні вимірювання атмосферного тиску.

3.2 Пояснення до програм, за якими працює мікропроцесорний пристрій, що проектується

Мікропроцесорний пристрій, що проектується, використовує дві основні програми для своєї роботи: Arduino IDE для програмування мікроконтролера та Android Studio для розробки мобільного додатку.

У середовищі Arduino IDE розроблюється програмне забезпечення для мікроконтролера, яке включає в себе код для зчитування даних з датчиків температури, вологості та тиску, їх обробки та подальшої передачі. Використовуючи Arduino IDE, ви можете взаємодіяти з бібліотеками для спрощення роботи з датчиками та іншими пристроями.

У той же час, в середовищі Android Studio розроблюється мобільний додаток, який взаємодіє з мікропроцесорним пристроєм. Цей додаток може отримувати дані від мікроконтролера через зв'язок , USB.

Android Studio дозволяє створювати інтерактивні та зручні для користувача інтерфейси, що дозволяють відображати та аналізувати дані з мікропроцесорного пристрою.

Таким чином, у моєжму проекті використовуються дві потужні інструментальні платформи, які дозволяють ефективно розробляти як

мікропроцесорне програмне забезпечення, так і мобільні додатки для взаємодії з ним.

ТЕСТУВАННЯ ТА ВІДЛАДКА ПРОГРАМИ

Створення тестових сценаріїв:

- Розроблення тестових сценаріїв для кожної функціональності програми, включаючи зчитування даних з датчиків, передачу даних на мобільний пристрій та їх відображення.
- Визначення очікуваних результатів для кожного тестового сценарію.

Тестування на етапі розробки:

- Виконання інкрементальних тестів під час написання коду для перевірки коректності функціональності на кожному етапі розробки.
- Відлагодження програми на ранніх етапах для виявлення та виправлення помилок.

Організація тестування:

- Запуск тестових сценаріїв на віддалених та локальних пристроях для перевірки взаємодії між Arduino та мобільним додатком.
- Використання емуляторів для тестування на різних моделях мобільних пристроїв та операційних системах.

Тестування функціональності:

- Перевірка правильності зчитування даних з датчиків та їх передача на мобільний пристрій.
- Впевненість у коректності обробки даних та їх відображення на екрані мобільного пристрою.
- Перевірка реакції програми на різні сценарії, включаючи зміну умов навколишнього середовища.

Відлагодження (дебагінг):

- Використання відлагоджувальних інструментів для виявлення та усунення помилок у програмному коді.
- Стеження за значеннями змінних та контроль виконання програми крок за кроком.
- Виправлення помилок у реальному часі для забезпечення стабільної та надійної роботи програми.

Тестування на реальних умовах:

- Перевірка програми на реальних умовах за різних погодних умов (температура, вологість, тиск).
- Врахування факторів зовнішнього середовища та їх впливу на роботу датчиків та програми.

Тестування екстрених ситуацій:

- Перевірка реакції програми на непередбачені ситуації, такі як втрата зв'язку, відсутність сигналу з датчиків тощо.
- Виявлення та усунення можливих проблем та збоїв у програмі.

4 Висновки по проекту

У результаті проектування та реалізації метеостанції на базі Arduino Nano з виводом даних з датчиків на телефон через USB-з'єднання, було досягнуто всіх поставлених завдань та отримано такі результати:

Придатність пристрою до виконання покладеного завдання

Розроблений пристрій повністю відповідає своєму призначенню - він здатен точно зчитувати та передавати дані про температуру, вологість та атмосферний тиск. Метеостанція забезпечує стабільний збір даних з датчиків DHT11 і BMP180 та передачу цих даних на мобільний пристрій, де вони відображаються у зручному для користувача вигляді. Така система є придатною для моніторингу кліматичних умов у різних середовищах, включаючи житлові приміщення, офіси, теплиці та промислові об'єкти.

Оцінка ефективності пристрою

Ефективність розробленої метеостанції підтверджується її здатністю надавати точні та своєчасні кліматичні дані. Використання доступних і надійних компонентів, таких як Arduino Nano, датчики DHT11 і BMP180, забезпечує високу точність вимірювань та стабільність роботи системи. Передача даних через USB-з'єднання гарантує швидкий та безперебійний обмін інформацією між метеостанцією та мобільним пристроєм.

Оцінка експериментальних результатів

Експериментальні результати підтверджують високу точність вимірювань, які здійснюються метеостанцією. Всі виміряні параметри - температура, вологість та тиск - відповідають реальним умовам навколишнього середовища та не мають значних відхилень. Це дозволяє зробити висновок про надійність та точність роботи пристрою, що робить його придатним для використання у різних практичних сценаріях.

Характеристики спроектованого пристрою

Основні характеристики розробленої метеостанції включають:

Контролер: Arduino Nano

Датчики:

- DHT11 для вимірювання температури та вологості
- BMP180 для вимірювання атмосферного тиску

Спосіб підключення: USB-з'єднання з мобільним пристроєм

Інтерфейс: Мобільний додаток для платформи Android, який відображає отримані дані у зручному вигляді

Точність вимірювань

- Температура: ±2°C

- Вологість: ±5% RH
- Атмосферний тиск: ±1 hPa

Загальні висновки

Проект "Метеостанція на базі Arduino Nano з виводом даних з датчиків на телефон" демонструє високу ефективність та надійність у зборі та передачі кліматичних даних. Вона є доступною для реалізації навіть для початківців у сфері електроніки та програмування, завдяки простоті використання компонентів та розробленого програмного забезпечення. Такий пристрій може знайти широке застосування у різних сферах життя, забезпечуючи користувачів необхідною інформацією про кліматичні умови в режимі реального часу.

СПИСОК ЛІТЕРАТУРИ

1. Історія появи метеостанцій [Електронний ресурс]: https://www.rusimpuls.ru/about/publications/4156/

2. www.integraengineering.in/blog/electromagnetic-relays;

3. www.electrical4u.com/electromagnetic-relay-working-types-ofelectromagnetic-relays/;

4. www.io.adafruit.com;

5. www.iotcircuithub.com/smart-home-with-google-assistant-alexa/;

6. www.instructables.com/google-assistant-controlled-switch-using-nodemcu/;
7. Datasheet ATmega168, 2016;

8. What are weather stations and why do you need it for weather forecasting [Електронний ресурс]: https://windy.app/blog/what-are-the-weather-stations.html

9. https://arduino.ua/ru/prod185-datchik-vlajnosti-i-temperatyri-dht11

10. https://autohome.org.ua/market/sensors/bmp180-detail

11. https://developer.android.com

12. ATMEGA328P Datasheet, PDF - Alldatasheet

Додаток А - Код Arduino IDE

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>
#include <DHT.h>

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);

void setup() {
  Serial.begin(9600);
  dht.begin();
  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP180 sensor, check wiring!");
    while (1);
  }
}
void loop() {
  delay(2000);

  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  sensors_event_t event;
  bmp.getEvent(&event);
  float pressure = event.pressure;

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C\t");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" hPa");

  Serial.print(temperature);
  Serial.print(",");
  Serial.print(humidity);
  Serial.print(",");
  Serial.println(pressure);
}
```

Додаток Б - Код Android Studio

MainActivity

```
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.hardware.usb.UsbDevice
import android.hardware.usb.UsbDeviceConnection
import android.hardware.usb.UsbEndpoint
import android.hardware.usb.UsbManager
import android.os.Build
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.util.Log
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat


class MainActivity : AppCompatActivity() {

    private lateinit var usbManager: UsbManager
    private var usbDeviceConnection: UsbDeviceConnection? = null
    private var usbDevice: UsbDevice? = null
    private var usbEndpoint: UsbEndpoint? = null

    private lateinit var connectionStatusTextView: TextView
    private lateinit var temperatureTextView: TextView
    private lateinit var humidityTextView: TextView
    private lateinit var pressureTextView: TextView
    private lateinit var weatherTextView: TextView
    private lateinit var errorTextView: TextView
    private lateinit var progressBar: ProgressBar
    private lateinit var refreshButton: Button
    private lateinit var selectDeviceButton: Button

    private val handler = Handler(Looper.getMainLooper())
```

```
    companion object {
        private const val ARDUINO_VENDOR_ID = 0x6790
        private const val CHANNEL_ID = "ARDUINO_CHANNEL"
        private const val NOTIFICATION_ID = 1
        private const val ACTION_USB_PERMISSION =
"com.example.meteo.USB_PERMISSION"
    }

    private val usbReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context?, intent: Intent?) {
            when (intent?.action) {
                UsbManager.ACTION_USB_DEVICE_ATTACHED -> {
                    findArduinoDevice()
                }
                UsbManager.ACTION_USB_DEVICE_DETACHED -> {
                    usbDeviceConnection?.releaseInterface(usbDevice?.getInterface
(0))
                    usbDeviceConnection?.close()
                    sendNotification("Arduino Disconnected", "Arduino device is
disconnected.")
                    connectionStatusTextView.text = "Arduino Disconnected"
                }
                ACTION_USB_PERMISSION -> {
                    synchronized(this) {
                        val device =
intent?.getParcelableExtra<UsbDevice>(UsbManager.EXTRA_DEVICE)
                        if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                            device?.apply {
                                usbDevice = this
                                openUsbDevice()
                            }
                        } else {
                            errorTextView.text = "Permission denied for device
$device"
                        }
                    }
                }
            }
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
        usbManager = getSystemService(USB_SERVICE) as UsbManager
        connectionStatusTextView = findViewById(R.id.connectionStatusTextView)
        temperatureTextView = findViewById(R.id.temperatureTextView)
        humidityTextView = findViewById(R.id.humidityTextView)
        pressureTextView = findViewById(R.id.pressureTextView)
        weatherTextView = findViewById(R.id.weatherTextView)
        errorTextView = findViewById(R.id.errorTextView)
        progressBar = findViewById(R.id.progressBar)
        refreshButton = findViewById(R.id.refreshButton)
        selectDeviceButton = findViewById(R.id.selectDeviceButton)

        createNotificationChannel()

        val filter = IntentFilter()
        filter.addAction(UsbManager.ACTION_USB_DEVICE_ATTACHED)
        filter.addAction(UsbManager.ACTION_USB_DEVICE_DETACHED)
        filter.addAction(ACTION_USB_PERMISSION)
        registerReceiver(usbReceiver, filter)

        refreshButton.setOnClickListener {
            readData()
        }

        selectDeviceButton.setOnClickListener {
            findArduinoDevice()
        }

        findArduinoDevice()
    }
```

```
    private fun findArduinoDevice() {
        val usbDeviceList = usbManager.deviceList
        usbDeviceList.values.forEach { device ->
            Log.d("MainActivity", "Device found: ${device.deviceName}")
            if (device.vendorId == ARDUINO_VENDOR_ID) {
                usbDevice = device
                if (usbManager.hasPermission(device)) {
                    openUsbDevice()
                } else {
                    val permissionIntent = PendingIntent.getBroadcast(
                        this, 0, Intent(ACTION_USB_PERMISSION),
PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
                    )
                    usbManager.requestPermission(device, permissionIntent)
                }
                return@forEach
            }
        }
    }

    private fun openUsbDevice() {
        usbDevice?.let { device ->
            val usbInterface = device.getInterface(0)
            usbEndpoint = usbInterface.getEndpoint(0)
            usbDeviceConnection = usbManager.openDevice(device)
            usbDeviceConnection?.claimInterface(usbInterface, true)
            sendNotification("Arduino Connected", "Arduino device is connected.")
            connectionStatusTextView.text = "Arduino Connected"
            startReadingData()
        }
    }
```

```
    private fun startReadingData() {
        Thread {
            val data = ByteArray(64)
            while (!Thread.interrupted()) {
                val length = usbDeviceConnection?.bulkTransfer(usbEndpoint, data,
data.size, 1000)
                if (length != null && length > 0) {
                    val dataString = String(data, 0, length,
Charsets.UTF_8).trim()
                    Log.d("MainActivity", "Received data: $dataString")
                    handler.post {
                        updateUI(dataString)
                    }
                } else {
                    handler.post {
                        errorTextView.text = "Failed to read data"
                    }
                }
                Thread.sleep(5000)
            }
        }.start()
    }

    private fun readData() {
        progressBar.visibility = ProgressBar.VISIBLE
        Thread {
            val data = ByteArray(64)
            val length = usbDeviceConnection?.bulkTransfer(usbEndpoint, data,
data.size, 1000)
            handler.post {
                progressBar.visibility = ProgressBar.GONE
                if (length != null && length > 0) {
                    val dataString = String(data, 0, length,
Charsets.UTF_8).trim()
                    updateUI(dataString)
                } else {
                    errorTextView.text = "Failed to read data"
                }
            }
        }.start()
```

```
    }

    private fun updateUI(dataString: String) {
        val data = dataString.split(",")
        if (data.size >= 4) {
            temperatureTextView.text = "Temperature: ${data[0]}"
            humidityTextView.text = "Humidity: ${data[1]}"
            pressureTextView.text = "Pressure: ${data[2]}"
            weatherTextView.text = "Weather:"
            errorTextView.text = ""
        } else {
            errorTextView.text = "Invalid data format"
        }
    }
    override fun onDestroy() {
        super.onDestroy()
        unregisterReceiver(usbReceiver)
        usbDeviceConnection?.releaseInterface(usbDevice?.getInterface(0))
        usbDeviceConnection?.close()
    }

    private fun sendNotification(title: String, message: String) {
        val intent = Intent(this, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent: PendingIntent = PendingIntent.getActivity(this, 0,
intent, PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE)

        val builder = NotificationCompat.Builder(this, CHANNEL_ID)
            .setSmallIcon(android.R.drawable.ic_dialog_info)
            .setContentTitle(title)
            .setContentText(message)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)

        with(NotificationManagerCompat.from(this)) {
            notify(NOTIFICATION_ID, builder.build())
        }
    }
```

```
    private fun createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val name = "Arduino Notifications"
            val descriptionText = "Notifications for Arduino device connection
status"
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(CHANNEL_ID, name, importance).apply
{
                description = descriptionText
            }
            val notificationManager: NotificationManager =
                getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            notificationManager.createNotificationChannel(channel)
        }
    }
}
```

activity_main.XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    android:theme="@style/Base.Theme.MyApplication"
    android:padding="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/connectionStatusTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Connection Status"
        android:textColor="@color/purple_700"
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/temperatureTextView" />
```

```
<TextView
    android:id="@+id/temperatureTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Temperature"
    android:textColor="@color/purple_700"
    android:textSize="24sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/connectionStatusTextView"
    app:layout_constraintBottom_toTopOf="@id/humidityTextView" />

<TextView
    android:id="@+id/humidityTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Humidity"
    android:textColor="@color/purple_700"
    android:textSize="18sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/temperatureTextView"
    app:layout_constraintBottom_toTopOf="@id/pressureTextView" />

<TextView
    android:id="@+id/pressureTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pressure"
    android:textColor="@color/purple_700"
    android:textSize="18sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/humidityTextView"
    app:layout_constraintBottom_toTopOf="@id/weatherTextView" />

<TextView
    android:id="@+id/weatherTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Weather"
    android:textColor="@color/purple_700"
    android:textSize="18sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/pressureTextView" />
```

```
    <Button
        android:id="@+id/refreshButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh"
        android:textColor="@android:color/white"
        android:backgroundTint="@color/purple_700"
        app:layout_constraintTop_toBottomOf="@id/weatherTextView"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/errorTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textColor="@android:color/holo_red_dark"
        android:textSize="16sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/refreshButton" />

    <Button
        android:id="@+id/selectDeviceButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Device"
        android:textColor="@android:color/white"
        android:backgroundTint="@color/purple_700"
        app:layout_constraintTop_toBottomOf="@id/errorTextView"
        app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```