ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метеостанція на базі мікроконтролера ESP32»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 – Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

_____ *Андрій ФЕЩЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-42
Андрій ФЕЩЕНКО
Ім'я, ПРІЗВИЩЕ
Керівник: Артем АНТОНЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент:
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерна інженерія

Ступінь вищої освіти бакалавр

Спеціальність 123 – Комп'ютерна інженерія

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
_____
«_____» _____ 2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Фещенко Андрій Іванович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Метеостанція на базі мікроконтролера ESP32,
керівник кваліфікаційної роботи Артем АНТОНЕНКО к.т.н., доцент ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56
2. Строк подання кваліфікаційної роботи «3» червня 2024р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література; мікроконтролер ESP32; датчик BME280; дисплей E-Ink 4.2"; акумулятор Li-Pol; мова програмування C++; Arduino IDE; бібліотеки: Adafruit BME280, GxEPD2, WiFi.h, HTTPClient.h, ArduinoJson; API OpenWeatherMap; протокол HTTP; JSON-структури; технологія deep sleep; бездротове з'єднання Wi-Fi.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз технічного завдання. Огляд існуючих рішень
2. Огляд літератури та аналогічних розробок
3. Технічне завдання. Вимоги та вибір
4. Апаратна частина проєкту
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2025	Виконано
2	Обробка матеріалу	23.03-7.04.2025	Виконано
3	Розробка теоретичної частини	08.04-14.04.2025	Виконано
4	Розробка електричної схеми	14.04-16.04.2025	Виконано
5	Висновки за результатами аналізу	17.04-20.04.2025	Виконано
6	Оформлення пояснювальної записки	20.04-08.05.2025	Виконано
7	Розробка презентації	18.05-27.05.2025	Виконано
8	Передзахист	27.05-29.05.2025	Виконано
9	Здача в деканат	30.05-05.06.2025	Виконано

Здобувач вищої освіти _____ Андрій ФЕЩЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Артем АНТОНЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 100 стор., 12 рис., 1 табл., 19 джерел.

Мета роботи – Розроблення метеостанції на базі мікроконтролера ESP32

Об'єкт дослідження – процес моніторингу кліматичних параметрів за допомогою сучасних апаратно-програмних засобів.

Предмет дослідження – метеостанція на базі ESP32 під назвою Paragon: ESP32 Weather Display, що дозволяє вимірювати температуру, вологість та атмосферний тиск з подальшою візуалізацією даних на енергоефективному дисплеї E-Ink.

Короткий зміст роботи: Кваліфікаційна робота приурочена розробці автономної метеостанції на базі мікроконтролера ESP32 де основною метою є створення компактного та енергоефективного пристрою для вимірювання основних кліматичних параметрів, таких як атмосферний тиск, температура та вологість, з можливістю отримання погодних даних з інтернету через API OpenWeatherMap.

У роботі проведено аналіз сучасних аналогів, обґрунтовано вибір апаратних компонентів таких як дисплей E-Ink на 4.2", датчик BME280, акумулятор Li-Pol та мікроконтролер ESP32 як обчислювального модуля. Здійснено розробку схеми з'єднання, апаратного середовища та програмного забезпечення з використанням потрібних бібліотек в середовищі Arduino IDE.

КЛЮЧОВІ СЛОВА: ESP32, BME280, E-Ink, Li-Ion, МЕТЕОСТАНЦІЯ, ТЕМПЕРАТУРА, ТИСК, ПОГОДА, МЕТЕОДАНІ.

ЗМІСТ

ПЕРЕЛІК ІСНУЮЧИХ РІШЕНЬ…….. 8
ВСТУП…… 9
1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ……... 10
1.1 Обґрунтування актуальності теми……... 10
1.2 Мета дослідження та основні завдання…… 11
1.3 Об’єкт дослідження та предмет дослідження…….. 11
1.4 Сучасні метеостанції на базі ESP32……..……. 12
1.5 Переваги використання E-Ink дисплеїв…….. 13
1.6 Аналіз проекту «Paragon: ESP32 Weather Display»…….. 14
2 ТЕХНІЧНЕ ЗАВДАННЯ. ВИМОГИ ТА ВИБІР…….….. 16
2.1 Функціональні вимоги……... 16
2.2 Вибір апаратної платформи…… 17
2.3. Визначення програмного забезпечення…… 18
2.4 Вимоги до користувацького інтерфейсу…… 20
2.5 Вимоги до надійності та експлуатації…… 22
3 ОБҐРУНТУВАННЯ ФУНКЦІОНУЮЧИХ ВИМОГ……... 25
3.1 Вибір мікроконтролера та його опис……...…..…… 25
3.2 Датчик BME280. Функції та підключення……. 30
3.3 E-Ink дисплей: особливості та інтеграція…… 33
3.4 Li-pol акумулятор. Специфікації та характеристики…… 37
3.5 Розробка схеми з'єднань……... 42
3.6 Збірка та монтаж системи……. 45
3.7 Налаштування апаратного середовища…….. 47
ВИСНОВКИ…… 54
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ……... 56
Додаток А. Програмний код …….. 59
Додаток Б. Демонстраційні матеріали(презентація)…… 94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ESP32 – Мікроконтролер на базі 32-бітного процесора Tensilica Xtensa LX6;

BME280 – Датчик температури, вологості та атмосферного тиску;

Li-Po – Літій-полімерний акумулятор;

E-Ink – Електронний папір (e-paper), дисплей на електронних чорнилах;

GxEPD2 – Бібліотека для роботи з дисплеями E-Ink;

I2C – Протокол послідовного інтерфейсу для підключення сенсорів;

SPI – Послідовний периферійний інтерфейс для підключення дисплеїв;

API – Інтерфейс прикладного програмування (Application Programming Interface);

JSON – JavaScript Object Notation — формат обміну даними;

WiFi – Бездротова технологія передачі даних;

HTTP – Протокол передачі гіпертексту;

OLED – Органічний світлодіодний дисплей;

FreeMonoBold9pt7b – Шрифт із бібліотеки Adafruit GFX;

HSRP, VRRP, GLBP – Протоколи FHRP;

IaaS, SaaS, PaaS – Моделі хмарних обчислень;

TCP, UDP, DHCP – Мережеві протоколи.

ВСТУП

Дослідження в даному проекті представляє цікавий підхід до інтеграції технологій в інтелектуальну метеостанцію. В останні роки країна зазнала спаду сільськогосподарської діяльності через зміну клімату, що спричиняє велику небезпеку для сільськогосподарського виробництва. Це дозволяє станції фактично прогнозувати стихійне лихо, використовуючи датчики температури та вологості які вимірюють температуру та вологість в атмосфері.

Ця система розроблена в першу чергу для використання на шкільних метеорологічних станціях, вона також застосовується на різних метеостанціях. Основною метою є створення недорогої інтелектуальної системи для зберігання даних, отриманих шляхом вимірювання різних фізичних параметрів в атмосфері, таких як температура та вологість. Мікроконтролер ESP-32 використовується як технологія для зберігання виміряних даних. Це сучасне та ефективне рішення для підключення до Інтернету. Система інтелектуальної метеостанції побудована навколо таких кроків: пряме зондування навколишнього середовища, вимірювання та зберігання даних, а також надання користувачам доступу до результатів через дисплей. Система побудована навколо набору вбудованих датчиків, мікроконтролера, що служить ядром системи, та протоколу бездротового інтернет-зв'язку. Результати відображаються в зрозумілому графічному вигляді на електронному чорно-білому дисплеї.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Обґрунтування актуальності теми

Щоб зрозуміти, чи актуальна ця тема, потрібно взглянути на сучасні проблеми які нас оточують. Погодне середовище, економічна криза з якою зв’язано підвищення цін на продукти та недостаток часу із-за праці. І завдяки розвитку технологій ці проблеми можливо вирішити за допомогою автономних пристроїв, що здатні автоматично збирати, аналізувати та відображати дані з навколишнього середовища і при більших ресурсах зробити майже повністю автономну ферму для вирощування овочів. Це є одне з найбільш можливих способів використання метеостанції, але її також можливо використовувати в побутових потребах або промислових. Також за допомогою метеостанції можливо зробити умний дім, завдяки якому автономно можливо підтримувати температуру в домі і рівень освтілюванності, що допоможе більш ефективніше використовувати і проводити свій час.

І завдяки появі недорогих, потужних і енергоефективних компонентів появляється можливість створювати компактну, автономну та енергоефективну метеостанцію на базі мікроконтролера ESP32 з використанням E-Ink дисплею завдяки якому можливо виводити інформацію з мінімізованним споживанням енергії, що актуально при використанні споживання від батареї.

Завдяки цім факторам можемо сказати, що тема кваліфікаційної роботи є актуальною, оскільки поєднує сучасні технології і сучасні можливості використання з подальшим розвитком проєкту.

1.2 Мета дослідження та основні завдання

Метою дослідження є розробка інформаційного прототипу метеостанції на базі міктроконтрлера ESP32, який буде показувати погоду та температуру заданого міста на дисплеї E-Ink, та температуру і погоду приміщення вокруг пристрою зібрану за допомогою датчика BME280.

Для досягнення мети необхідно виконати ряд завдань:

– Провести аналіз існуючих рішень у сфері IoT пристроїв для моніторингу погоди;

– Обдумати вибір апаратної частини для проєкту;

– Скласти електричну схему підключенням компонентів прототипу;

– Створити програмний код прототипу для отримання даних про погоду за допомогою API OpenWeatherMap, збирання локальних даних з датчика BME280 та обробки і відображення даних на E-Ink дисплеї;

– Зібрати прототип та протестувати його;

– Оцінити стабільність роботи та точність даних для зручного користування;

– Провести аналіз можливого удосконалення та використання приладу в різних умовах.

1.3 Об’єкт дослідження та предмет дослідження

Об’єктом дослідження знаходяться мікропроцесорні системи призначені для автоматизованого збору інформації, її обробки та відображення на дисплеї стан про навколишнє середовище. Особливу увагу виділено пристроям на базі мікроконтролера ESP32, які поєднують в собі можливості бездротової отримання та передачі даних, високої ефективності та енергоефективності. Системи такого типу використовуються у сфері інтернету речей, де необхідно забезпечити автономну роботу і безперервне отримання даних. У контексті проекту іде вивчення особливостей побудови автономного погодного дисплею, який здатен без

втручання отримувати локальні дані з датчика вимірювання даних навколишнього середовища, так і прогноз погоди через інтернет.

В цьому досліджені охоплюється як і апаратна частина, так і програмна реалізація цієї роботи. Також завдяки цьому дослідженню вивчається принцип взаємодії між мікроконтролером, датчиком і E-Ink дисплеєм завдяки яким є можливість оптимізації енергоспоживання за рахунок паперового дисплею, і також способи роботи з API-сервісами для отримання прогнозу погоди. Особливим предметом уваги є забезпечення стабільної та зручної візуалізації показників на дисплей для комфортного використання пристрою в побутовому середовищі.

1.4 Сучасні метеостанції на базі ESP32

З розвитком технологій, мікроконтролери з власним процесором на прикладі ESP32 отримали безліч застосувань, в тому числі і метеостанції. В цьому мікроконтролері висока обчислювальна потужніть, вбудовані модулі WI-FI та Bluetooth і також є підтримка багатьох периферійних інтерфейсі, завдяки чому цей мікроконтролер є ідеальною основою для побудови автономної погодної системи.

Сучасні метеостанції які створені на базі мікроконтролера ESP32 мають компактні розміри та можливість підключення різних датчиків, завдяки чому мають низьке енергоспоживання та отримання розширеної інформації про погоду при підключені до інтернету. В більшості випадків в таких станціях використовують датчики температури, тиску та вологості BME280 або DHT22, модуль годинника реального часу і також дисплеї для виведення інформації, прості LCD або сучасні E-Ink дисплеї.

Особливу популярність мікроконтролер ESP32 отримав за допомогою активного використання серед розробників DIY-проєктів. На різних платформах таких як GitHub, Hackster.io та інших форумах можливо знайти багато різних відкритих проєктів серед яких будуть і метеостанції які виконують подібні функції, такі як локальний моніторинг показників клімату, побудови графіків, надсилання

даних у хмарне сховище та надсилання повідомлень користувачеві. Також є проекти, які взаємодіють і керують іншими пристроями на основу погодних даних отриманих з метеостанції, наприклад автоматичне поливання рослин, овочів або відкриття жалюзі коли наступив ранок і встало сонце.

1.5 Переваги використання E-Ink дисплеїв

E-Ink дисплеї стали популярним рішенням в DIY проєктах і в тому числі компактних метеостанціях завдяки своїм перевагам. Головною перевагою таких дисплеїв є неймовірна енергоефективність, так як вони споживають струм лише в момент оновлення зображення. Проста метеостанція з таким дисплеєм може працювати від невеликої батареї місяцями, завдяки чому така метеостанція є ідеальним варіантом для автономного використання в різних галузях. Ця перевага є важливою для проектів з мікроконтролером ESP32, де енергозбереження не завадить.

Ще однією важливою перевагою є дуже добра читабельність при будь якому освітленні навколо на відміну звичайних LCD-екранів, на яких майже нічого непомітно на сонці. При прямому сонячному світлі E-Ink дисплей залишається чітким і контрастним, що дає можливість встановити пристрій біля вікна або на відкритому повітрі. Завдяки відсутності підсвічування ці дисплеї є комфортними при тривалому спостережені без напруги зору.

Також важливою характеристикою є стабільна робота в широкому діапазоні температур. Такі дисплеї можуть працювати як в спекотну погоду, так і у сильний мороз, що відрізняє їх від багатьох інших типів дисплеїв. Це дозволяє використовувати їх в метеостанції без додаткового захисту від клімату.

Однак варто враховувати і деякі мінуси цих дисплеїв. Наприклад, швидкість оновлення E-Ink дисплеїв є багато нижчою порівняно з LCD-екранами, за рахунок чого робить їх менш придатними для відображення більш динамічної інформації. Більшість доступних моделей є монохромними, хоч і з’являются кольорові

варіанти їх вартість значно вища за рахунок чого не користується популярністю.

В сучасних проектах E-Ink дисплеї довели свою ефективність в сумісності з мікроконтролерами по типу ESP32. Завдяки цій зв’язці є можливість створювати компактні та енергоощадні пристрої з великим терміном роботи, що важливо для домашніх метеостанцій працюючих цілодобово. Проекти продовжують вдосконалюватись, і в майбутньому можуть появитись ще більш досконалі рішення для відображення метеоданих.

1.6 Аналіз проекту «Paragon: ESP32 Weather Display»

Дана персональна метеостанція є відкритим проектом, яка привернула увагу своєю простотою та ефективністю. Автор хотів створити енергоефективний пристрій з мінімальним споживанням електроенергії, яке досягається за рахунок використання E-Ink дисплею та ретельного програмування режимів сну мікроконтролера.

Ідея приладу полягає в створені абсолютно автономного пристрою, який здатний працювати від невеликого компактного акумулятора протягом тривалого часу. Дані про погоду система отримує як від локальних датчиків температури, вологості та тиску, так і з інтернет джерел за рахунок WI-FI підключення. Особливістю проєкта є те, що мікроконтролер активізується лише при оновлені даних на дисплеї на короткий час і після чого переходить в режим глибокого сну.

Реалізація такого проєкту включає використання практичних та доступних компонентів таких як:

- Мікроконтролер FireBeetle ESP32-E;
- Датчик BME280 для вимірювання метеоданих;
- E-Ink дисплей 7.5”;
- Літій-полімерний акумулятор.

Програмний код був написаний на платформі Arduino IDE з використанням спеціальних бібліотек для роботи з дисплеєм та датчиками. Мікроконтролер

проводить менше 1% часу у активному стані, так як особливу увагу було приділено алгоритмам енергозбереження.

Проект зацікавлює тим, що демонструє можливості його створення з мінімальними витратами, але водночас має певні обмеження у вигляді відсутності захисту від вологості при використанні на вулиці, обмежену частоту оновлення даних та простий інтерфейс без додаткових функцій.

2 ТЕХНІЧНЕ ЗАВДАННЯ. ВИМОГИ ТА ВИБІР

2.1 Функціональні вимоги

Розробка метеостанції на базі мікроконтролера ESP32 повинно виконувати ключові функції спрямовані на моніторинг погодних умов в реальному часі. Основна увага спрямована на створення компактного та енергоефективного пристрою який здатен працювати в автономному режимі із використанням відкритих інтернет ресурсів, що дозволить отримувати прогноз погоди на найближчий час так і фактичні вимірювання кліматичних параметрів.

Головною задачею метеостанції є точне вимірювання основних кліматичних параметрів такі як температура повітря в діапазоні від -20°C до +60°C, відносну вологість та атмосферний тиск з забезпеченням максимальної автономності пристрою. Для ціх цілей використовується комбінація даних з інтернет джерел за допомогою OpenWeatherMap Api та локального датчику BME280, що забезпечить непогану достовірність вимірювань, а також надання актуального прогнозу погоди без необхідності постійного моніторингу ресурсів користувачем.

Важливим складовим елементом є відображення інформації на енергоощадному E-Ink дисплеї, за допомогою якого можливо виводити з високою контрастністю інформацію, не залежно від освітлення, зберігаючи виведену інформацію без постійного споживання енергії. Використовуючи інтервали при оновленні інформації, котрі можливо встановити від 15 до 60 хвилин, можливо забезпечити компроміс між точністю та тривалістю автономної роботи, за рахунок чого можливо мінімізувати частоту активації модулів та дисплею.

Пристрій розрахований на автономну роботу від Li-ion акумулятора з можливістю підзарядки. Для мінімізації енергоспоживання потрібно реалізувати режим глибокого сну між оновленнями даних та оптимізувати роботу з WI-FI підключенням.

2.2 Вибір апаратної платформи

Це є одним із ключових етапів створення проєкту будь-якої вбудованої системи, так як саме від технічних можливостей обраного мікроконтролера залежить стабільність роботи з функціональністю та потенціалом розширення можливостей пристрою. В межах даного проєкту був обраний мікроконтролер ESP32, який поєднує в собі велику кількість виводів вводу-виводу, підтримку сучасних інтерфейсів за допомогою яких можливо з легкістю підключити багато датчиків, сенсорів, дисплеї та інших периферійних пристроїв, вмонтований модуль WI-FI та Bluetooth та потужний двоядерний процесор.

Однією з головних причин вибору даного мікроконтролеру є саме його здатність працювати в режимах низького енергоспоживання, що дає змогу значно продовжити термін автономної роботи проєкту, що дуже важливо для метеостанції, яка повинна оновлювати дані з великими часовими інтервалами без потреби постійного енергоживлення. Крім того, даний мікроконтролер має вбудовані модулі підключення як до бездротової мережі WI-FI, так і до пристроїв з технологією Bluetooth без використання зовнішніх комунікаційних модулів.

Ще додатковою перевагою є широка програмна підтримка та дуже активна спільнота розробників різних DIY проєктів. Велика кількість технічної документації, кількість прикладів та бібліотек, які значно полегшують процес розробки для створення проєкту, так як це знижує ризик помилок і забезпечить гнучкість у майбутньому розришені функціоналу. Так же ESP32 сумісний із популярними середовищами розробки такі як Arduino IDE та PlatformIO, що робить його більш зручним мікроконтролером навіть для менш досвідчених людей, такі як студенти.

Крім переваг в функціоналі, непоганою складовою при виборі ESP32 є його доступність на ринку та відносно невисока вартість, за рахунок чого ця платформа є привабливою не лише для створення прототипу, але й для можливого масштабування проєкту. У цієї плати є велика кількість модифікацій та моделей – з дисплеями, з вбудованими антенами, акумуляторними контролерами тощо, що

дає можливість обрати ту саму конфігурацію, яка більше відповідає вимогам конкретному майбутньому проєкту.

Окремої уваги заслуговує сумісність ESP32 із широким спектром зовнішніх компонентів. Для створення функцій метеостанції важливо забезпечити коректну та стабільну роботу з датчиком BME280, а також підтримку інтерфейсу для E-Ink дисплею. Даний мікроконтролер чудово справиться з цим завданням, забезпечивши належний рівень швидкодії та точності при збереженні низького енергоспоживання.

2.3 Визначення програмного забезпечення

Ефективне використання метеостанції неможливе без ретельно спроєктованого програмного забезпечення, яке зможе забезпечити зв’язок між апаратними компонентами і допоможе забезпечити обробку даних, зручну візуалізацію результатів та оптимізацію енергоспоживання. Для реалізації даного проєкту програмна частина повинна виконувати такі важливі задачі, як ініціалізація та взаємодія з периферійними пристроями, отримувати погодні дані з інтернет мережі через API, оновляти графічний інтерфейс на дисплеї, керування режимом сну, обробка даних з датчиків, а підключення до мережі за допомогою WI-FI.

Основним середовищем для розробки програмного забезпечення використовується Arduino IDE, оскільки воно забезпечує легку роботу з ESP32 і має підтримку широкого спектру бібліотек, а також має велику спільноту, що легко дозволяє знайти приклади коду та вирішення технічних проблем. Для отримання прогнозу погоди використовується REST API від сервісу OpenWeatherMap, доступ до якого буде реалізовуватись через HTTP запити, які обробляє безпосередньо мікроконтролер. Для роботи з мережею використовуватиметься вбудована бібліотека WiFi, за допомогою якої мікроконтролер матиме змогу підключатись до заданої точки доступу та здійснювати обмін даними з сервером.

Програмне забезпечення також повинно забезпечити взаємодію з датчиком

BME280 при використанні інтерфейсів мікроконтролера. Отримані дані вологості, тиску та температури перед обробкою перевіряються на коректність і у випадку відсутності нових даних застосовуються останні збережені значення. Графічне відображення даних буде здійснюватися за допомогою спеціалізованих бібліотек для E-Ink дисплеїв, що дасть змогу створити читабельний та структурний інтерфейс з мінімальним використанням ресурсів мікроконтролера.

Один з головних моментів в програмному забезпеченні приділено реалізації глибокого сну для енергозбереження. Після завершення циклу оновлення даних пристрій переходить у режим глибокого сну, в якому енергоспоживання знижується до мінімуму. Вихід з режиму глибокого сну здійснюється за допомогою таймера, який налаштований на заданий інтервал і після чого сценарій програми повторно виконує цикл зчитування, обробки та виводу інформації.

У процесі розробки програмного коду потрібна оптимізація роботи з пам’яттю, так як ESP32 має дуже обмежений обсяг оперативної пам’яті і в парі з E-Ink дисплей який потребує обробки графічних буферів, потрібно забезпечити стабільність системи в умовах тривалого та безперервного використання, необхідно правильно використовувати динамічну пам’ять, своєчасно звільняти ресурси після виконання завдання та уникати надмірного копіювання великих об’єктів.

Додатковим аспектом є забезпечене стабільне з’єднання до інтернет-ресурсів. Для цього буде створено механізм перевірки наявності інтернет підключення для повторного з’єднання при втраті сигналу. Додатково забезпечено контроль над часом очікування відповіді від серверів, що дозволяє уникати зависання мікроконтролера у випадках нестабільної роботи інтернет мережі або відсутності доступу до API. При неможливості отримати дані з інтернету, система залишатиме і використовуватиме останні отримані значення, щоб забезпечити неперервність роботи пристрою.

Для збереження часу та синхронізації з інтернет даними використовуються дані із сервісів або внутрішній годинник мікроконтролера, які автоматично враховують часову пояси. За рахунок чого з’являється можливість точно

регулювати інтервали оновлення дисплея та таймер переходу в режим глибокого сну. Також буде реалізована програмна можливість налаштування частоти оновлення даних відповідно до потреб користувача або конкретних умов використання.

Програмний код структуровано у вигляді модульної архітектури, в якому функціональні блоки такі, як обробка даних, оновлення дисплея, робота з API, керування живленням та зчитування сенсорів – ізольовані в окремі частини. Такий підхід спростить налаштування та дасть змогу швидко вносити зміни і забезпечить гнучкість при масштабуванні або адаптації системи до інших задач. Завдяки відкритості та поширеності інструментів які використовуються у проєкті, програмне забезпечення може легко бути легко вдосконалене у майбутніх версіях пристрою або повторно використане в інших проєктах із подібними деталями.

2.4 Вимоги до користувацького інтерфейсу

Користувацький інтерфейс є критично важливою складовою для сучасного електронного пристрою, тому що саме він забезпечує взаємодію користувача з системою. В випадку даної метеостанції, яка базується на мікроконтролері ESP32 та E-Ink дисплеї, користувацький інтерфейс виконує функцію подання інформацію про прогноз погоди у зручному, зрозумілому та візуально привабливому вигляді. Тому при його розробці є як і функціональні вимоги, так і ергономічні вимоги, які мають забезпечити комфортну експлуатацію приладу незалежно від технічних знань користувача.

Однією з основних вимог є інформативність інтерфейсу, який повинен чітко відображати основні метеорологічні параметри такі, як відносна вологість, температуру повітря, поточні погодні умови, а також атмосферний тиск та прогноз погоди на найближчі години або дні. Вся інформація повинна бути згрупована та структурована за логічними блоками. У верхній частині екрану, наприклад, може відображатись поточний час та дата, а нижче – температура та інші кліматичні

параметри, а в ніжній частині – тенденції зміни погоди чи графічні піктограми прогнозу погоду. Це дозволяє користувачу миттєво орієнтуватись у показниках, не втрачаючи час на розшифрування значень чи символів.

Важливим критерієм є читабельність, що дає вибраний E-Ink дисплей, який має додаткові особливості такі, як чорно-була кольорова палітра, відсутність підсвічування та інерційність оновлення зображення. З одного боку, це сприяє на зручність читання при денному світлі, а з іншого потребує уважного підходу до елементів інтерфейсу таких, як шрифти, розмір тексту, товщина ліній та розташування елементів. Інтерфейс має уникати зайвих графічних деталей, що не несуть корисної інформації та бути достатньо контрастним. Для основних числових значень краще використовувати великі та жирні шрифті, тоді як для другорядних елементів можливо використовувати більш менший розмір. Також потрібно мінімізувати кількість оновлень екрану, так як часте перемальовування E-Ink дисплею впливає на підвищення енергоспоживання та його ресурс.

Крім візуального наповнення, інтерфейс повинен мати логічну структуру, що означає, що дані повинні відображатись в послідовності, яка відповідає природному сприйняту, а саме спочатку ідуть актуальні значення, потом прогноз погоди і після чого іде допоміжна інформація. Для позначення погодних умов, а саме сонячна, хмарна, дощ тощо, бажано використовувати прості графічні піктограми, які будуть зрозумілі без пояснень. Надписів повинно бути по мінімуму, лише там, де це необхідно для унеможливлення неоднозначного тлумачення.

Також важливо є зробити універсальність та незалежність від мови. У випадку проєктування пристрою для широкого кола користувачів слід враховувати, що не всі можуть розуміти текст певної мови інтерфейсу. І в цьому випадку будуть використовуватися загальноприйняті скорочення такі, як «°C», «%», «hPa» та символи, які інтернаціонально зрозумілі.

В інтерфейсі також повинна бути адаптація до зміни даних, що означає, що за відсутності інтернет з’єднання або сигналу від датчика пристрій повинен не просто зависати, а й інформувати користувача про тимчасову недоступність даних і показувати останні отримані значення. Така поведінка сприяє довірі користувача

до пристрою та демонструє надійність системи. Також, у разі помилок з нестачею заряду батареї або помилок з API можливо передбачити відповідні повідомлення у нижній частині дисплею або у вигляді спрощених іконок.

У контексті вимог до інтерфейсу потрібно розглянути питання енергоефективності. Оскілки дисплей оновлюється з певним інтервалом, необхідно програмно реалізувати логіку, за допомогою якої можливо уникати зайвих оновлень у разі, коли значення параметрів не змінилися не змінились або не змінились суттєво. При зміні температури лише на 0.1°C немає потреби перемальовувати весь екран. Даний підхід значно продовжить термін автономної роботи приладу, що є критичним у польових умовах чи при використанні в місцях з обмеженим доступом до джерел живлення.

Також варто звернути увагу на можливість конфігурації інтерфейсу на кодовому рівні, щоб залишити можливість вносити зміни в розташування блоків, змінювати шрифти, перемикати інтервали оновлення або додавати нові функції такі, як графіки змін температури за добу для досвідчених користувачів або розробників. Таке програмне налаштування під себе підвищує привабливість пристрою для розширеного кола користувачів.

2.5 Вимоги до надійності та експлуатації

Розробка електронних пристроїв, особливо які призначені для тривалої автономної роботи в реальних умовах середовища, потребує врахування цілого комплексу до їхньої надійності та експлуатаційної стійкості. Метеостанція на базі ESP32 має працювати без постійного обслуговування, забезпечуючи стабільне отримання, обробку та відображення кліматичних параметрів протягом тривалого часу. Так що ще на етапі проєктування необхідно передбачити заходи, які забезпечать довговічність приладу, коректне функціонування при змінних зовнішніх умовах, його безпеку, а також зручність експлуатації для користувача.

Усі компоненти системи повинні бути підібрані з урахуванням їхньої

стійкості до зовнішніх впливів таких, як температурні коливання, пил, вібрації, вологість. Мікроконтролер ESP32, як головний компонент системи, показав високу стабільність роботи в промислових та побутових умовах, але його ефективність експлуатації значною мірою залежить від правильності охолодження, рівня живлення, монтажу та захисту від електростатичних розрядів. Так само важливо обрати якісні датчики, які також сертифіковані для тривалого використання на відкритому повітрі або в не опалювальних приміщеннях.

Також ще одним важливим аспектом є енергетична надійність системи, оскільки пристрій повинен працювати в автономному режимі на акумуляторі, то йому потрібен захист від перенапруги, коротких замикань, нестабільності струму та перерозряду. Акумуляторна батарея повинна бути якісною та з високим ресурсом циклів зарядки-розрядки. Система живлення має мати стабілізатор напруги, який дозволить ESP32 та периферії працювати при змінному рівні заряду. Наявність індикатора рівня заряду або програмної функції контролю живлення значно підвищить надійність пристрою та дозволить уникнути критичних відмов.

Програмна стійкість також грає важливу роль, так як мікроконтролер повинен мати захист від збоїв у роботі прошивки, такий, як наприклад програмний «watchdog» таймер, який перезапускає систему у разі зависання пристрою, що дозволить уникнути ситуацій, коли прилад повністю вийде із ладу через дрібну помилку в алгоритмах. Програма повинна мати продуману логіку обробки виключень і також перевірку доступності до мережі, працездатності датчиків та відповідей API.

Також варто приділити увагу експлуатаційній простоті та зручності обслуговування пристрою. Оскільки метеостанція не передбачає щоденної взаємодії та налаштувань від користувача, то її корпус має бути герметичним або напівгерметичним для захисту від вологи, пилу та комах, але при цьому забезпечувати прости доступ до кнопки перезавантаження, акумулятора і USB-роз’єму. Також є бажаною функцією можливість оновлення прошивки по повітрю, що дозволить виправляти баги або додавати нові функції без необхідності розбирання пристрою.

Не менш важливим аспектом ще є екологічна стійкість, так як пристрій може використовуватися у приміщеннях де висока вологість, або на улиці, теплицях, навісами і також на дачах без постійного нагляду. Тому матеріал корпусу повинен мати стійкість до ультрафіолету, не піддаватися швидкому старінню і мати захищені кабельні з’єднання від окислення. Датчик BME280 повинен бути розташований таким способом, щоб забезпечити доступ повітря і також уникати конденсації вологи або потрапляння прямих крапель.

Також систему потрібно захистити від зовнішніх загроз, таких як стрибки напруги, збої Wi-Fi мережі, спроби несанкціонованого доступу або порушення живлення. Для цього можливо використовувати апаратні фільтри живлення, а в прошивці методи валідації відповіді від серверів та базові елементи безпеки мережі, таких як збереження паролів у безпечному вигляді, шифрування трафіку, використання SSL - з’єднання до OpenWeatherMap.

Ще потрібно враховувати зовнішні механічні впливи по типу вібрацій, ударів та можливого падіння пристрою. У таких випадках варто застосовувати амортизуючі прокладки всередині корпусу або закріплення плати на стійних опорах. Сам корпус має бути міцним і виконаним з не ламкого пластику або кріпкої деревини, якщо пристрій буде використовуватись в екстремальних умовах.

З експлуатаційної тривалості, розрахунковий термін служби метеостанції має бути не менше 3-5 років без серйозного технічного обслуговування чи серйозних поломок. Досягти такого тривалості можливо за рахунок як надійних комплектуючих, так і за продуманої стратегії енергозбереження та правильної терморегуляції, захисту від зовнішніх факторів та мінімізації рухомих або вразливих компонентів.

3 ОБҐРУНТУВАННЯ ФУНКЦІОНУЮЧИХ ВИМОГ

3.1 Вибір мікроконтролера та його опис

Ключовим елементом будь-якого мікропроцесорного пристрою є його обчислювальний модуль у вигляді мікроконтролера. І тому в межах даного проєкту було обрано мікроконтролер ESP32, який є високопродуктивною бездротовою мікросхемою з інтегрованими модулями бездротового з’єднання Wi-Fi та Bluetooth, який розроблений компанією Espressif Systems. Цей вибір був обумовлений багатьма факторами, які включають в себе енергоефективність, наявність периферійних інтерфейсів, продуктивність, підтримку сучасного програмного забезпечення, можливість бездротового з’єднання та надійність у роботі в автономному режимі.

Мікроконтролер ESP32 має два ядра Xtensa LX6 з тактовою частотою 240 МГц та 520 КБ оперативної пам’яті, що дозволяє виконувати паралельні обчислення, що є важливим для обробки даних з кількох датчиків одночасно і може працювати як в режимі стану готовності, так і в режимі сну з дуже низьким споживанням енергії. Від підтримує різні протоколи зв’язку, такі як Bluettoth, Wi-Fi, BLE і Ethernet. Він має вбудований USB-порт, що дозволяє програмувати і взаємодіяти з мікроконтролером через комп’ютер. Також на платі є широкий спектр GPIO-виходів для підключення різної периферії таких як I2C, SPI, UART та ADC, які дозволяють підключати різні датчики, наприклад як датчики вологості, температури, тиску та інші компоненти системи.

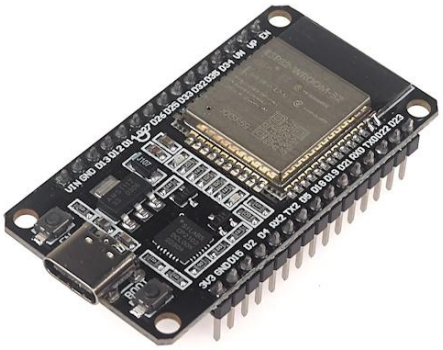


Рисунок 1 – Мікроконтролер ESP32

Для розробки даного проєкту було обрано варіант ESP32-WROOM-32, який широко підтримується в екосистемі Arduino та має достатній обсяг пам’яті для буферизації даних та зберігання прошивки. Забезпечити можливість інтеграції з великою кількістю зовнішніх пристроїв і датчиків, включно з метеорологічним датчиком BME280 і дисплеєм E-Ink можливо за допомогою таких характеристик, який має даний мікроконтролер:

– 520 КБ SRAM;
– До 16 МБ флеш-памяті;
– Апаратну підтримку Wi-Fi 802.11 b/g/n і Bluetooth 4.2;
– Великий набір цифрових та аналогових інтерфейсів, включаючи SPI, I2C, UART, ADC, PWM;
– Напругу живлення 3.3 В;
– USB-UART чіп CH340;
– Wi-Fi стандарти FCC/CE/IC/TLEC/KCC/SRRC/NCC;
– Частотний діапазон 2.4 ~ 2.5 ГГц;
– Радіо NZIF приймач з чутливістю -98dBm;
– Діапазон робочих температур -40 ~ +85 °C;
– Режими Wi-Fi Station/softAP/SoftAP+station/P2P;
– Захист WPA/WPA2/WPA2-Enterprise/WPS;

– Шифрування AES/RSA/ECC/SHA;

– Оновлення ПЗ: UART Download / OTA (по мережі) / download and write firmware via;

– Мережеві протоколи: IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT;

– Розміри плати 28x51 мм;

– Відстань між контактними пінами: 25 мм;

– Вага 10 г.

ESP32 має 36 виводів, які можуть бути використані для підключення додаткових елементів, таких як датчики, екрани, різні модулі та інші периферійні пристрої. Деякі з цих виводів можуть також використовуватися для зв’язку з мікроконтролерами, сенсорними екранами та іншими пристроями через інтерфейси. Самі виводи мікроконтролеру ESP32 мають такі значення:

– GPIO 1 - 5 - Вхід / вихід загального призначення;

– GPIO 12 - 16 - Вхід / вихід загального призначення;

– 3.3V - Вихід напруги 3,3 В;

– GND – Земля;

– EN - Ввімкнення / вимкнення контролера ESP32;

– IO0 - Режим програмування / робочий режим;

– MTDO - Вихід даних SPI Flash;

– MTCK - Вхід клоку SPI Flash;

– U0R - Вхід для UART0, RXD;

– U0T - Вихід для UART0, TXD;

– IO25 - Вхід / вихід загального призначення;

– IO26 - Вхід / вихід загального призначення;

– IO27 - Вхід / вихід загального призначення;

– IO32 - Вхід / вихід загального призначення;

– IO33 - Вхід / вихід загального призначення;

– VIN - Вхід напруги від 5 В до 12 В.

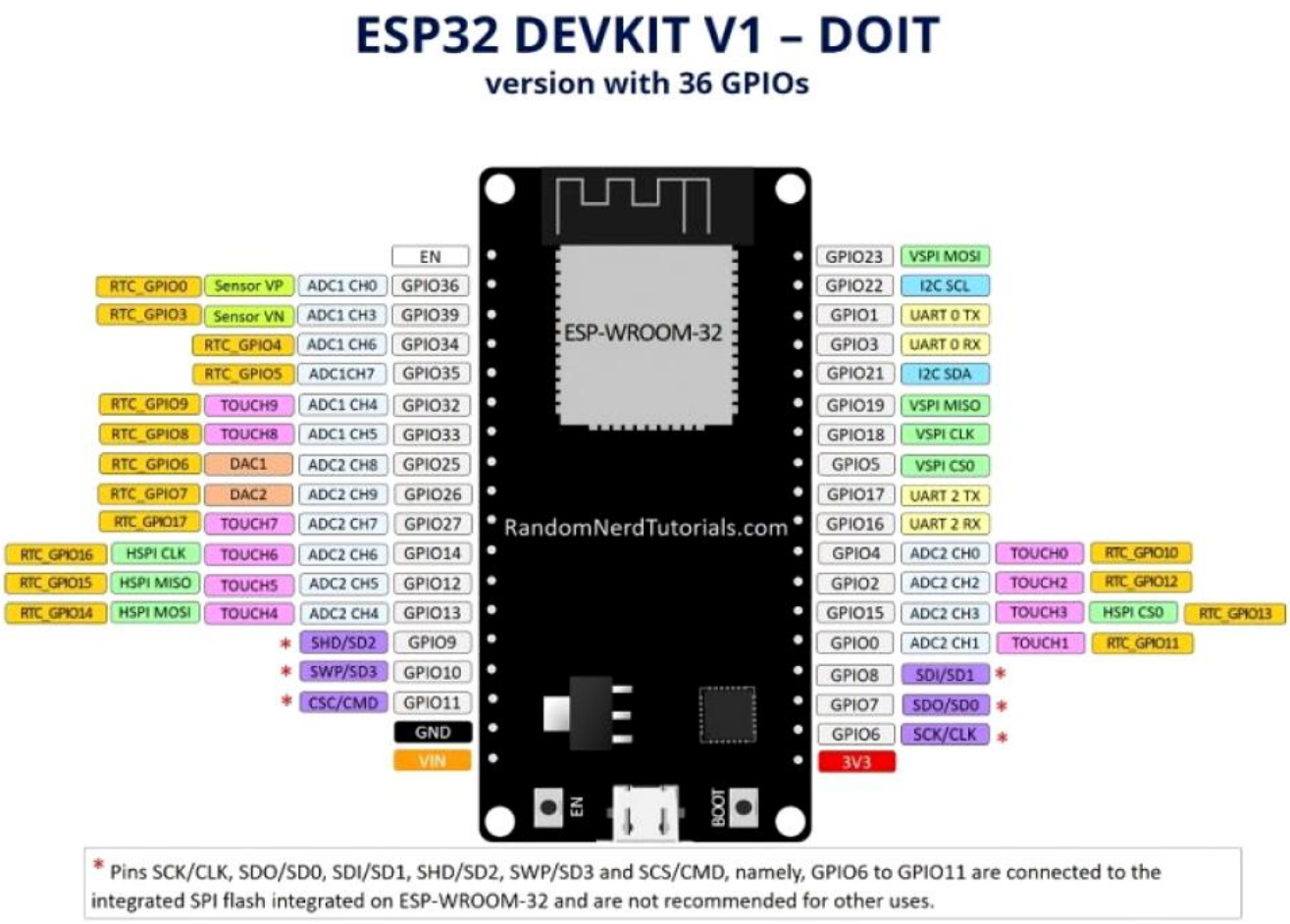


Рисунок 2 – Розпіновка ESP32

У контексті завдання створити енергоефективну та автономну метеостанцію, що отримує дані з інтернету та локальних датчиків, обмінюється інформацією через Wi-Fi і відображає її на дисплеї сприяли дані фактори:

– ESP32 має розвинені можливості енергозбереження, такі як підтримка режимів сну, зокрема Deep Sleep, що дозволяє зменшити споживання енергії до мікроамперів у стані очікування, що є важливим для пристрою, який працює від акумулятора. У рамках метеостанції мікроконтролер більшість часу перебуватиме в режимі глибокого сну, прокидаючись лише для оновлення даних і виводу їх на дисплей.

– Наявність вбудованого Wi-Fi модулю суттєво спростить реалізацію підключення до інтернет мережі, усуваючи необхідність використовувати

додаткові плати або модулі. Це дозволить зменшити габарити пристрою, загальне енергоспоживання і його вартість.

– Мікроконтролер ESP32 має достатньо обчислювальну потужність для одночасного виконання кількох завдань, таких як формування запитів до OpenWeatherMap, обробка кліматичних даних, парсинг отриманої інформації, підготовка графіки для дисплея та керування живленням. Багатоядерність дозволяє реалізувати багатозадачність яка знижує час реакції системи та дозволяє уникнути зависань.

На відміну від більшості плат Arduino, рідною напругою ESP32 DevKit є 3,3, а не 5 В. Виходи для логічної одиниці видають 3,3 В, а в режимі входу очікують приймати не більше 3,3 В. Висока напруга може пошкодити мікроконтролер.

Кожен пін введення-виведення платформи підтримує апаратні інтерфейси, нижче в таблиці 1.

Таблиця 1 – Призначення інтерфейсів виводу

Інтерфейс	Кількість	Призначення
I²C	2	Використовується для спілкування з периферією за послідовним інтерфейсом «I²C».
SPI	3	Для спілкування з периферією за послідовним інтерфейсом SPI.
UART/Serial	3	Для спілкування з периферією за інтерфейсом UART.
I²S	2	Використовується для передачі та отримання цифрового звуку з іншими аудіопристроями.

Ця плата має всі необхідні на собі елементи, а саме стабілізатор живлення, піни для підключення зовнішніх модулів, USB-UART міст для програмування, кнопки Reset та Boot, світлодіодні індикації. Вона дуже добре підходить для

створення прототипу, а згодом може бути замінена на більш компактне рішення або інтегрована в друковану плату пристрою.

3.2 Датчик BME280. Функції та підключення

В системах моніторингу погодних умов ключову роль відіграють датчики вимірювання, які відповідають за точне вимірювання фізичних параметрів навколишнього середовища. В цьому проєкті використовується датчик BME280 який являється високоточним цифровим сенсором, розроблений компанією Bosch Sensortec, який вимірює атмосферний тиск, температуру повітря і відносну вологість. Його компактність, висока точність вимірювання та низьке енергоспоживання роблять його ідеальним вибором для побудови автономної метеостанції на базі мікроконтролера ESP32.

Датчик BME280 є вдосконаленою версією попередніх моделей BMP180 та BMP280, що поєднує в собі три незалежні сенсорні модулі в одному корпусі, а сампе барометр, гігрометр і термометр, за рахунок чого він має такі характеристики:

- Температурний діапазон від -40°C до +85°C;
- Точність вимірювання температури ±0.5°C;
- Діапазон вимірювання вологості від 0% до 100% RH;
- Точність вологості ±3% RH;
- Діапазон атмосферного тиску від 300 до 1100 гПа;
- Точність тиску ±1 гПа;
- Інтерфейси підключення I2C або SPI;
- Напругу живлення від 1.8 В до 3.6 В;
- Струм в режимі вимірювання тиску: 714 мкА;
- Струм в режимі вимірювання вологості: 340 мкА;
- Споживаний струм в режимі вимірювання температури: 350 мкА;
- Струм в режимі сну: від 0.1 мкА до 0.5 мкА;

– Розміри корпусу близько 15 х 12 х 3 мм.

Датчик має цифровий інтерфейс, за рахунок чого легко зчитувати дані за допомогою мікроконтролера без використання аналогових перетворювачів. В даному випадку використовується I2C протокол, який дозволяє задіяти лише два сигнальні дроти для передачі даних, а саме SDA та SCL, що зменшить складність схемі і кількість необхідних пінів ESP32.

BME280 працює за принципом комбінації MEMS-датчиків, кожен з яких виконує свою функцію, а саме:

– *Гігрометричний сенсор*. Полімерний елемент який змінює свою ємність залежно від вологості повітря;

– *Температурний сенсор*. Кремнієвий термістор який змінює опір у залежності від температури. Показники температури також використовуються для корекції інших вимірювань;

– *Барометричний сенсор*. Тензометричний елемент який змінює електричний опір при зміні тиску, що дозволяє оцінити як абсолютний тиск, так і зробити прогноз на зміну погодних умов.

За допомогою комбінованих показників можливо точно оцінити не лише локальні кліматичні умови, а й прогнозувати зміни погоди, виявляти тренди в коливаннях вологості та тиску, що є цінним для довготривалого моніторингу навколишнього середовища.

Серед багатьох датчиків які доступні для вимірювання кліматичних параметрів, BME280 був обраний за такими критеріями як:

– Точність та стабільність яка забезпечує надійні показники навіть при зміні зовнішніх умов;

– Універсальність яка об’єднує в собі три сенсори в одному корпусі, що дозволяє спростити конструкцію пристрою;

– Мінімальне енергоспоживання датчика який має декілька режимів роботи, включаючи наднизький енергоспоживальний режим, що дозволяє застосовувати його в автономних пристроях;

– Простота інтеграції за допомогою широкої підтримки в Arduino-сумісних

бібліотеках, що полегшує розробку прошивки.

Також датчик BME280 має декілька режимів:

– Sleep Mode – енергозберігаючий режим, де всі вимірювання зупинені;

– Forced Mode – датчик виконує одне вимірювання при виклику команди, а після повертається в режим сну;

– Normal Mode – датчик періодично виконує вимірювання та зберігає результати в регістрах.

У нашому випадку найбільше підходить режим Forced Mode, так як пристрій більшість часу перебуватиме в режимі глибокого сну, а після пробудження виконує одноразове зчитування даних.

Також датчик BME280 можливо використовувати не лише для базового моніторингу, а й для розширених задач по типу:

– Оцінка висоти над рівнем моря за барометричним тиском;

– Обчислення точки роси на основу температури та вологості;

– Адаптація дисплея при зміні графіки залежно від умов.



Рисунок 2 – Датчик BME280

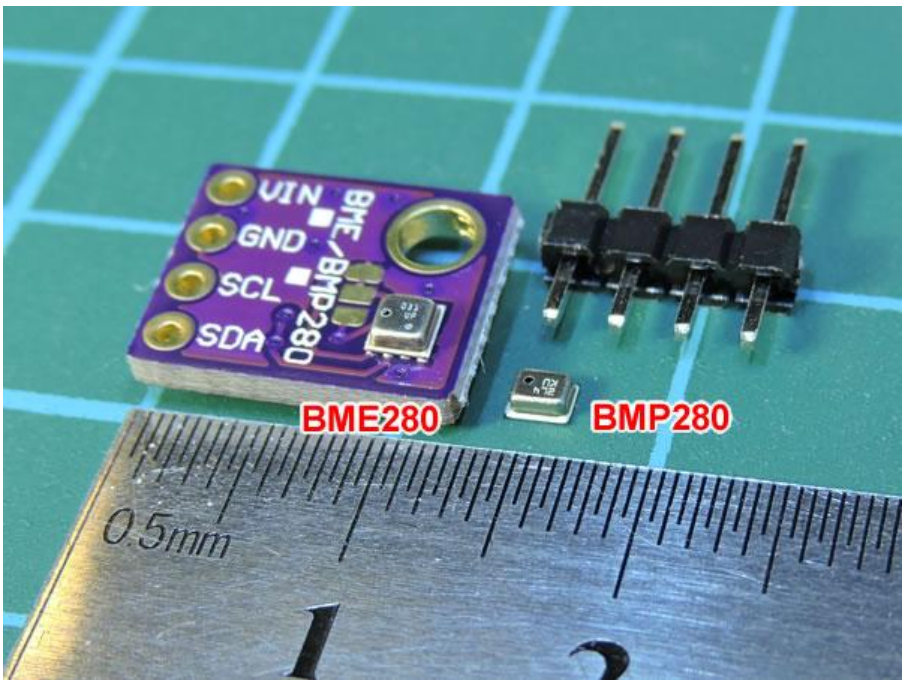


Рисунок 3 – Порівняння датчиків BME280 та BMP280

3.3 E-Ink дисплей. Особливості та інтеграція

Одним з ключових елементів розумної метеостанції, яка впливає на енергоефективність, візуальне сприйняття даних та зручне використання є дисплей. В даному проєкту використовується E-ink дисплей на електронних чорнилах, що є сучасною технологією відображення інформації, що значно відрізняє його від традиційних рідкокристалічних або OLED дисплеїв.

E-Ink або Electronic Ink – це тип електрофоретичного дисплея, який імітує вигляд звичайного чорнила на папері. Кожен піксель складається з мікрокапсул, у яких знаходяться заряджені білі та чорні частинки і залежно від прикладеної напруги, ті або інші частинки рухаються до поверхні, створюючи зображення.

Основною перевагою даного дисплею є те, що енергія споживається лише під час оновлення зображення, після чого дисплей зберігає статичне зображення без жодних енергозатрат, що ідеально підходить для метеостанції, де оновлення інформації виконується з інтервалом у 15-60 хвилин.

Перевагами даного E-Ink дисплею є:

– Чіткість зображення, коли навіть при яскравому сонячному світлі дисплей залишається повністю читабельним, що важливо при використанні пристрою на вулиці;

– Низьке енергоспоживання є найбільш критичним параметром для автономного пристрою, завдяки тому, що дисплей не потребує живлення для підтримки зображення і він може працювати довше без підзарядки;

– Компактні розміри та легкість інтеграції дисплею за допомогою різних форм-факторів, які легко підключаються до мікроконтролерів ESP32 за допомогою SPI-інтерфейсу;

– Комфорт для очей за рахунок відсутності підсвітки екрану робить читання з такого дисплею схожим на читання з паперу.

У проєкті використовується E-Ink дисплей діагоналлю 4.2 дюйми з роздільною здатністю 400x300 пікселів, що дозволяє виводити:

– Атмосферний тиск;

– Температуру;

– Вологість;

– Піктограми для іконок погоди;

– Прогноз погоди з OpenWeatherMap;

– Дату, час і рівень заряду батареї.

Саме поняття E-Ink перекладається як "електронний папір" або "електронне чорнило". Дисплеї, створені з урахуванням технології E-ink називають електронним папером - т.к. Ця технологія розроблялася саме для імітації стандартного друку на звичайному аркуші паперу. Унікальність технології "рідке чорнило" полягає в мікро-капсулах, заповнених мікроскопічними гранулами чорного та білого кольору. При подачі електричного струму до такої капсули – гранули розподіляються з її боків залежно від полярності напруги. Т.о. сторони капсули змінюють свій колір.

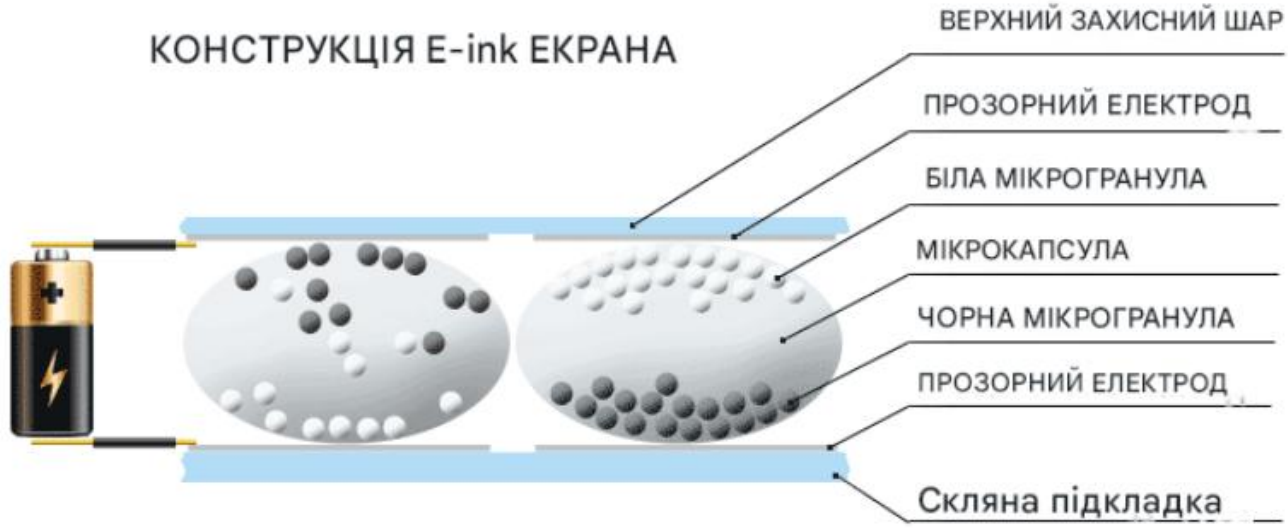


Рисунок 4 – Конструкція E-Ink дисплею

Недоліками таких дисплеїв є:

– Відсутність білого "білого" кольору, так як він насправді, відображається трохи сіруватим;

– Висока статичність оновлення зображень на таких дисплеях відбувається набагато повільніше в порівнянні з тими ж TFT-дисплеями, що робить, наприклад, неможливим перегляд динамічного контенту (але, заради справедливості потрібно помітити, що "електронний папір" і не створювався для цих цілей). А от зате ті ж електронні книги працюють місяцями з такими дисплеями без підзарядки.

Донедавна E-ink дисплеї були монохромні та низькоконтрастні. Але технології на місці не стоять і якість зображення відчутно зросла на сьогоднішній день - картинка виглядає просто приголомшливо.

Оскільки дисплей оновлюється повільніше, ніж LCD або OLED, зазвичай 1 – 3 секунди, то необхідно мінімізувати кількість оновлень. У системі реалізований механізм відкладеного оновлення, а саме оновлення дисплея відбувається лише раз на заданий інтервал часу, що дозволить:

– Уникати мерехтіння;
– Зберігати високу стабільність зображення;
– Економити заряд акумулятора.

Дисплей має також можливість часткового оновлення, що може використовуватись для швидшого виводу динамічної інформації такої як годинник,

хоча це обмежено специфікаціями конкретного модуля.

Для роботи з дисплеєм у середовищу Arduino IDE потрібно використовувати бібліотеки такі як:

– *GxEPD2*. Одна з найпопулярніших бібліотек з підтримкою великої кількості моделей дисплеїв;

– *Adafruit GFX*. Бібліотека для виводу тексту, графіки і фігур, яку можно поєднувати з GxEPD2.

Бібліотеки дозволяють легко створити інтерфейс користувача з температурними шкалами, логотипами, піктограмами погоди тощо. Також є можливість використовувати кастомні шрифти та виводити зображення у форматі BMP або XBM.

Обрана модель дисплею WeActStudio 4.2” є оптимальною за цими значеннями:

– Енергоефективність;
– Простота інтеграції з ESP32;
– Наявністю підтримки в бібліотеках;
– Розміром для перегляду інформації з відстані.



Рисунок 5 – Зовнішній вигляд дисплею спереду



Рисунок 6 – Зовнішній вигляд дисплею позаду

3.4 Li-pol акумулятор. Специфікації та характеристики

Живлення це один із найважливіших аспектів автономної роботи будь-якого мікроконтролерного пристрою. У контексті розробки енергоефективної метеостанції на базі ESP32 вибір джерела живлення напряму вливає на тривалість автономної роботи, безпеку експлуатації та стабільність системи. У даному проєкті як основне джерело живлення використовується Li-Ion акумулятор, що забезпечує необхідну ємність, компактність та багаторазову перезаряджуваність.

Літій-полімерний акумулятор — це різновид літій-іонного електричного акумулятора, який відрізняється від класичного літій-іонного акумулятора типом використаного у ньому електроліту. У випадку з Li-ion — це гелеподібний або рідкий електроліт, у випадку з Li-Pol — спеціальний щільний полімер, насичений розчином літію.

Напруга окремого LiPo елемента залежить від його хімічного складу і коливається від приблизно 4,2 В (повністю заряджений) до приблизно 2,7—3,0 В (повністю розряджений), де номінальна напруга становить 3,6 В або 3,7 В

(приблизно середнє значення найвищого та найнижчого значення) для елементів на основі оксидів літію-металу (наприклад, LiCoO2).

Для порівняння: літій-залізо-фосфатний(LiFePO4) елемент від 3,6—3,8 В (повністю заряджений) до 1,8—2,0 В (повністю розряджений).

Точні номінали напруг роботи елемента повинні бути вказані в специфікації продукту, що дозволить правильно спроектувати електронну схему, яка буде запобігати перезаряджанню або повному розряджанню під час його використання.

Акумуляторні батареї з LiPo елементами, з'єднаними послідовно та паралельно, мають окремі контакти для кожного елемента. Спеціалізований зарядний пристрій може контролювати заряд кожного елемента, щоб усі елементи були заряджені до однакової напруги - мали однаковий стан заряду (SOC)..

Більшість сучасних пристроїв забезпечені акумуляторами на літієвої основі, літій-іонними або літій-полімерними, тому надалі мова буде йти саме про них і про принципи їх роботи в акумуляторному інструменті. Характеристики літій-іонних акумуляторів принаймні в кілька разів перевищують аналогічні показники нікель-кадмієвих та нікель-метал-гідридних акумуляторів, які використовувалися раніше. Вони виробляють приблизно вдвічі більше енергії на питому одиницю ваги, ніж попереднє покоління удосконалених нікель-метал-гідридних (NiMH) акумуляторів.

Літій-іонні акумулятори (Li-ion) добре характеризують себе при роботі на великих токах, що важливо і необхідно при використанні. Вони мають досить низький саморозряд (1,5 - 3% на місяць), значну ємність і терміни служби, але вимагають дуже жорсткого слідування певним правилам експлуатації.

Умовно, ці правила можна розділити на дві групи:

– Які не залежать від користувача;

– Залежать від користувача.

До першої групи входять основні правила заряду і розряду акумуляторів, які контролюються вбудованим в акумулятор або безпосередньо в електроніку управління інструментом, BMS захисного контролера. Захисна схема, не допускає заряду-розряду нижче точки повернення і контролює температуру елемента. Однак

навіть ця схема не зможе захистити акумулятор, якщо ви залишите батарею повністю розрядженою на тривалий час. У першої групи є такі правила використання:

1. В силу хімічних особливостей, елемент (ячейка) літієвого акумулятора все своє життя повинен перебувати в стані, при якому його напруга не перевищує 4,2 вольта і не опускається нижче 2,7 вольта. Такі напруги є показниками максимального і мінімального робочого заряду.

2. Кількість енергії, що віддається акумулятором при його розряді від 4,2 до 2,7 вольт - це є його ємність. За допомогою вбудованої електроніки, виробники іноді обмежують максимальну і мінімальну напругу, що продовжує життя самого акумулятора, але при цьому його загальна ємність знижується.

3. Якщо з якихось причин, відбувається перезаряд або перерозряд, і напруга на акумуляторі виходить за межі зазначені вище, термін його життя драматично зменшується. Такі процеси є дуже згубними для акумулятора.

4. Вбудовані в акумуляторний блок BMS контролери, відстежують напруга на кожному елементі (ячейці) акумулятора і контролює температуру. Тому у справному стані контролер ніколи не дозволяє під час заряду напрузі стати більше 4,2 вольт, і опуститися нижче мінімального розряду 2,7 вольта. У деяких моделях інструментів Bosch, в самих акумуляторних блоках, для зменшення їх ціни, плати управління можуть бути відсутні. У такому випадку ці функції обов'язково виконує контролер в зарядному пристрої та в електроніці управління електроприладами. В акумуляторних інструментах преміум серії, щоб зберегти характеристики і тривалу працездатність акумуляторів, можуть використовувати одночасно обидва варіанти контролера.

Друга група, це правила експлуатації, на які безпосередньо впливає сам користувач, тим самим збільшуючи або зменшуючи термін життя акумуляторів. Розглянемо більш докладніше:

1. Потрібно намагатися уникати частих повних розрядів літієвих батарей. Не доводити акумулятор до мінімального заряду, а також до стану, коли обладнання саме вимикається, ну а якщо таке сталося не залишайте батарею на довго

розрядженою. Всупереч думці, що склалася, Li-ion акумулятори, на відміну від NiMH і NiCd не володіють "ефектом пам'яті", тому так звані "тренування" нового літієвого акумулятора не має сенсу. Li-ion акумулятор готовий до роботи вже з першого підключення.

2. Не потрібно боятися частих підзарядок, в тому числі і часткових, коли повний заряд не досягається. Літій-іонному акумулятору це не шкодить. В середньому термін експлуатації коливається від 500 до 1000 робочих циклів заряду і розряду. Надалі ємність знижується до 80% від максимальної. При великих навантаженнях і підвищенні робочих температур понад 40 ° С градусів, зниження ємності може статися вже після 300-500 циклів. Циклами вважаються тільки повні 100% заряд і наступний розряд до 0%. Наприклад, заряджаючи акумулятор при його 50-процентному розряді, буде "зарахована" тільки половина циклу, і втрата ємності відбудеться пізніше.

3. Перезаряд шкодить літієвим акумуляторам не менш, а навіть більше, ніж глибокий розряд. Слід пам'ятати, що незважаючи на те що контролер контролює максимальний рівень заряду, є один нюанс. Добре відомо, що напруга і ємність акумуляторів залежить від температури. Так, якщо, наприклад, ми зарядили акумулятор при кімнатній температурі і отримали заряд 100%, то при виході на мороз і його охолодженні, ступінь зарядженості акумулятора може знизитися до 80% і нижче. Але може бути і зворотна ситуація. Акумулятор, заряджений при низькій температурі до 100%, і будучи нагрітий до кімнатної температури, підніметься в напрузі і стане зарядженим, умовно кажучи до 105%, а це для нього дуже несприятливо.

4. Категорично забороняється заряджати літієві акумулятори не призначеними для цього зарядними пристроями без систем захисту і контролю температури. Це може призвести до вибуху і займання.

5. Літієві акумулятори старіють, навіть якщо не використовуються, а просто лежать на полиці. Це неминучий процес, і до того-ж досить сильно залежить від умов використання та зберігання. Ідеальні умови для тривалого зберігання акумулятора, це перебування поза устаткування з зарядом приблизно 50-60% і

плюсовою температурою 5-8 ° С. При повному заряді і температурі вище кімнатної, процес старіння прискорюється в 2-3 рази.

6. Не допускається надмірний нагрів акумулятора, наприклад, в результаті безперервних і тривалих навантажень. При нагріванні понад 70 ° С, відбувається руйнування іон-провідного захисного шару, металізація літію та інші несприятливі хімічні процеси, як наслідок швидке скорочення терміну служби, вихід акумулятора з ладу.

7. Літіеві акумулятори дуже не люблять заморожування. При низьких температурах ефективність будь-яких акумуляторів різко падає. У той час як NiCd продовжують працювати до -40 ° С, акумулятори NiMH і Li-ion при заморожуванні елементів до -20 ° С, припиняють функціонувати. Слід пам'ятати, що при мінусовій температурі, заряджати Li-ion акумулятори заборонено. Зберігання Li-ion акумуляторів при температурі нижче -6 ° С може вивести їх з ладу.

Для даного проєкту застосовується Li-ion акумулятор з номінальною напругою 3.7 В та ємністю 1000 мАг. Конкретний вибір залежав від вимог до тривалості автономної роботи та габаритних обмежень пристрою. Даний акумулятор має такі характеристики:

– Номінальна напруга 3.7 В;

– Максимальна напруга при повному заряді 4.2 В;

– Мінімальна доступима напруга розряду 2.5 – 3.0 В;

– Ємність 1000 мАг;

– Типом роз’єму є контактне кріплення за допомогою припаяних двух проводів.



Рисунок 7 – Li-pol акумулятор

Причинами вибору саме цього акумулятора є:

– Ємність акумулятор має непогану ємність за свою ціну, що дозволяє забезпечити багатоденну, а іноді і багатотижневу автономну роботу пристрою при правильній оптимізації споживання;

– Даний тип акумулятора є одним із найпоширенішим та доступним на ринку, що спрощує закупівлю та заміну;

– Фізична сумісність завдяки розмірам акумулятора є одним з непоганих факторів для використання в малогабаритних пристроях і також для компактної інтеграції у корпус метеостанції;

– Циклічність в даних акумуляторах непогана, вони витримають понад 500-1000 циклів заряджання без значної витрати ємності.

3.5 Розробка схеми з’єднань

Однією з ключових стадій розробки електронного пристрою є побудова правильної схеми з’єднань між усіма компонентами. В даному проєкту, в метеостанції на базі мікроконтролера ESP32 використовується чотири основні

елементи, такі як:

– Мікроконтролер ESP32;

– Датчик BME 280;

– 4.2” E-ink дисплей від компанії WeActStudio;

– Li-pol акумулятор.

Схема повинна забезпечувати такі принципи:

– Стабільне живлення всіх компонентів;

– Безпечне заряджання акумулятора;

– Коректну передачу даних через I2C для BME280 та SPI для E-Ink дисплею;

– Можливість переходу ESP32 у режим глибокого сну без впливу на інші модулі.

Зв’язок ESP32 з датчиком BME280 за допомогою I2C виконуватиме так:

– VCC > 3.3V на ESP32;

– GND > GND ESP32;

– SDA > GPIO 21;

– SCL > GPIO 22.

А зв’язок E-ink дисплею з мікроконтролером ESP32 через SPI за таким принципом:

– VCC > 3.3V ESP32;

– GND > GND ESP32;

– DIN(MOSI) > GPIO 23;

– CLK(SCK) > GPIO 18;

– CS(Chip Select) > GPIO 5;

– DC(Data/Command) > GPIO 17;

– RST(Reset) > GPIO 16;

– BUSY > GPIO 4.

ESP32 підтримує одночасну роботу через I2C та SPI, що дозволяє паралельно підключити BME280 та E-Ink дисплей без необхідності в мультиплексорах. Для I2C важливо, щоб обидва пристрої працювали на однаковій логіці – 3.3 В. Аналогічно, що E-Ink дисплей повинен бути моделью, яка підтримує логіку 3.3 В.

Також необхідно враховувати особливості дисплея, що він споживає енергію лише під час оновлення зображення, тому краще було б керувати живленням дисплея через транзистор або mosfet для повного вимкнення у період сну.

В графічному варіанті схеми, який буде додано до кваліфікаційної роботи, необхідно зобразити:

– ESP32 з усіма підключеними пінами
– BME280 підключений через I2C
– E-ink дисплей з SPI підключенням і керуючими сигналами
– Акумулятор.

Електричну схему було вирішено створювати в програмному забезпечені Fritzing яке забезпечить легке і ефективне проєктування схеми для метеостанції. Також нижче буде наведено інтерфейс даного програмного забезпечення на рисунку 8.

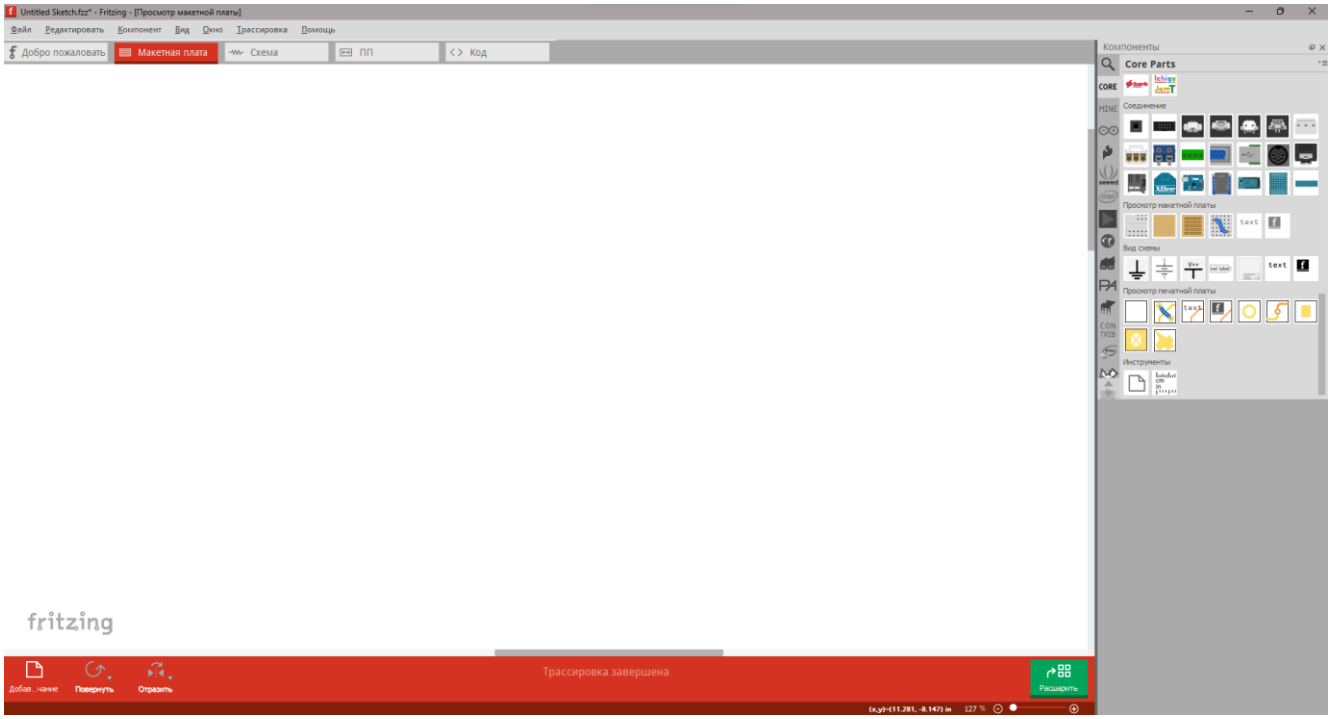


Рисунок 8 – Програмне забезпечення Fritzing

Нижче, на рисунку 9 буде зображено ретельно спроєктовану та реалізовану схему з’єднань створену в програмному забезпечені Fritzing, яка забезпечить

правильну роботу метеостанції в автономному режиму без збоїв і перевантажень з врахуванням електричних характеристик усіх компонентів і особливостей їх взаємодії, що є ключом для створення енергоефективної, стабільної та довговічної системи.

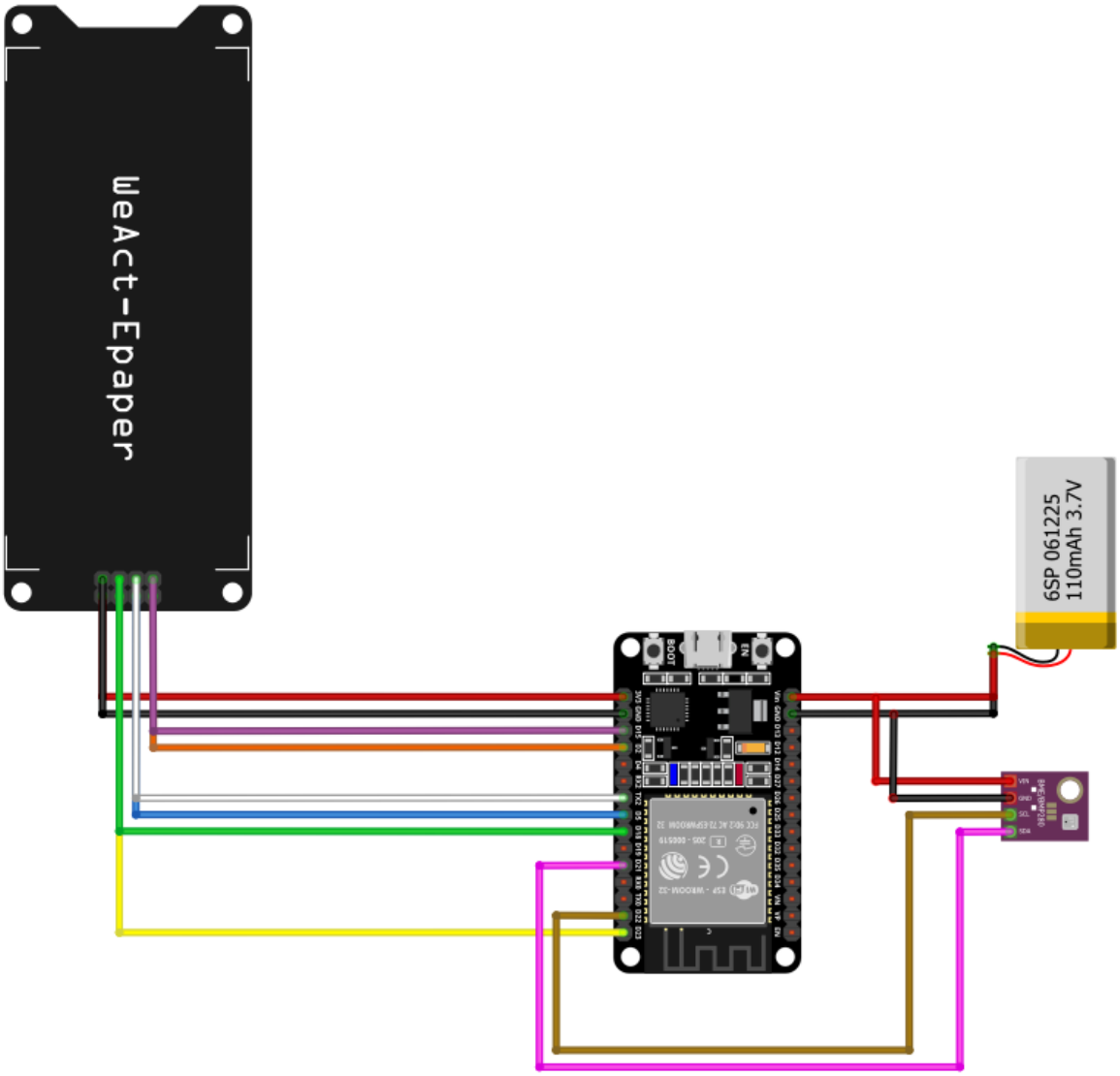


Рисунок 9 – Схема з’єднань

3.6 Збірка та монтаж системи

Збірка метеостанції є одні із завершальних та надзвичайно важливих етапів реалізації проєкту, так як цей єтап впливає на стабільність, довговічність та безпеку

його роботи в умовах автономного функціонування. Після підготовких всіх електронних компонентів і створення схеми з’єднань, необхідно перейти до фізичного з’єднання всіх елементів та тестування.

Монтаж електронних компонентів буде проходити в декілька етапів:

– Підключення дисплея за допомогою пінів, так як це прототип і це самий важливий елемент проєкту, то він в прототипній версії буде підключений за допомогою пінів замість пайки.

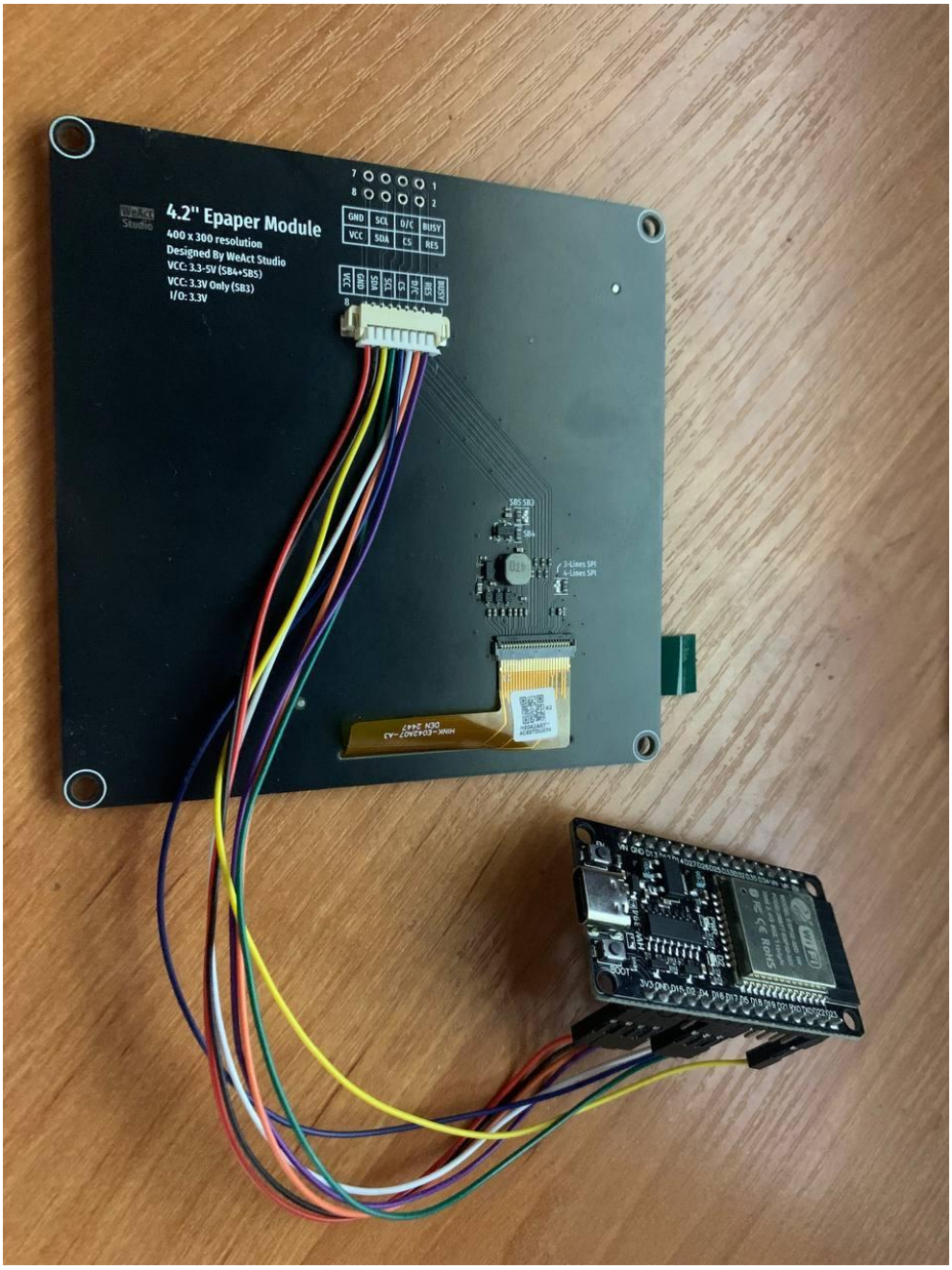


Рисунок 10 – Підключення дисплею за допомогою пінів

– Пайка датчику BME280 та Li-pol акумулятора до мікроконтролеру ESP32 буде відбуватись за допомогою паяльника, припою та тонкого багатожильного проводу у термоусадці і після надійного з’єднання місце паяння буде захищене термоклеєм.

– Закріплення всіх елементів прототипу на одній плоскості з міцного матеріалу за допомогою термоклея для тестування системи.

Після завершення монтажу всіх компонентів необхідно провести остаточне тестування пристрою на функціонування:

– Перевірити відображення даних на дисплеї;

– Виміряти споживання струму в активному режимі та в режимі глибокого сну;

– Протестувати оновлення даних з OpenWeatherMap API;

– Оцінити якість сигналу Wi-Fi.

Налаштування апаратного середовища передбачає поетапну підготовку та тестування кожного елементу системи, починаючи від написання та компіляції прошивки, та закінчуючи коректною роботою з дисплеєм, датчиком та мережевими сервісами. Грамотна реалізація цього етапу дозволить забезпечити стабільну, ефективну та автономну роботу метеостанції в реальному середовищі експлуатації.

3.7 Налаштування апаратного середовища

Налаштування апаратного середовища є критично важливим та завершальним етапом створення метеостанції, так як від цього залежить стабільність, енергоефективність та функціональність пристрою. Налаштування охоплює підготовку середовища розробки, налаштування мікроконтролера ESP32, перевірку з’єднання щ периферійними пристроями такі як BME280, E-Ink дисплеєм і Li-pol акумулятором, а також налагодження мережевого з’єднання та реалізацію режимів енергозбереження.

Для створення проекту було вибрано середовище програмування Arduino

IDE із-за своєї практичності і доступності при роботі з платами Arduino і в том числі ESP32 яку було використано для проекту. Arduino IDE – це програмне середовище розробки, що використовує C++ і призначене для програмування всіх плат Arduino. Абревіатура IDE розшифровується як Integrated Development Environment, у перекладі – інтегроване середовище розробки. За допомогою даного середовища можливо просто та легко написати програмний код для майже будь якого проєкту, за рахунок чого дана програма отримала таке широке використання та популярність серед DIY розробників.

Arduino IDE дозволяє складати програми для зручного текстового редактора, компілювати їх у машинний код, і завантажувати на всі версії плати Arduino. Програма є повністю безкоштовною, а скачати її можливо на офіційному сайті Arduino без якихось проблем любому бажаючому.

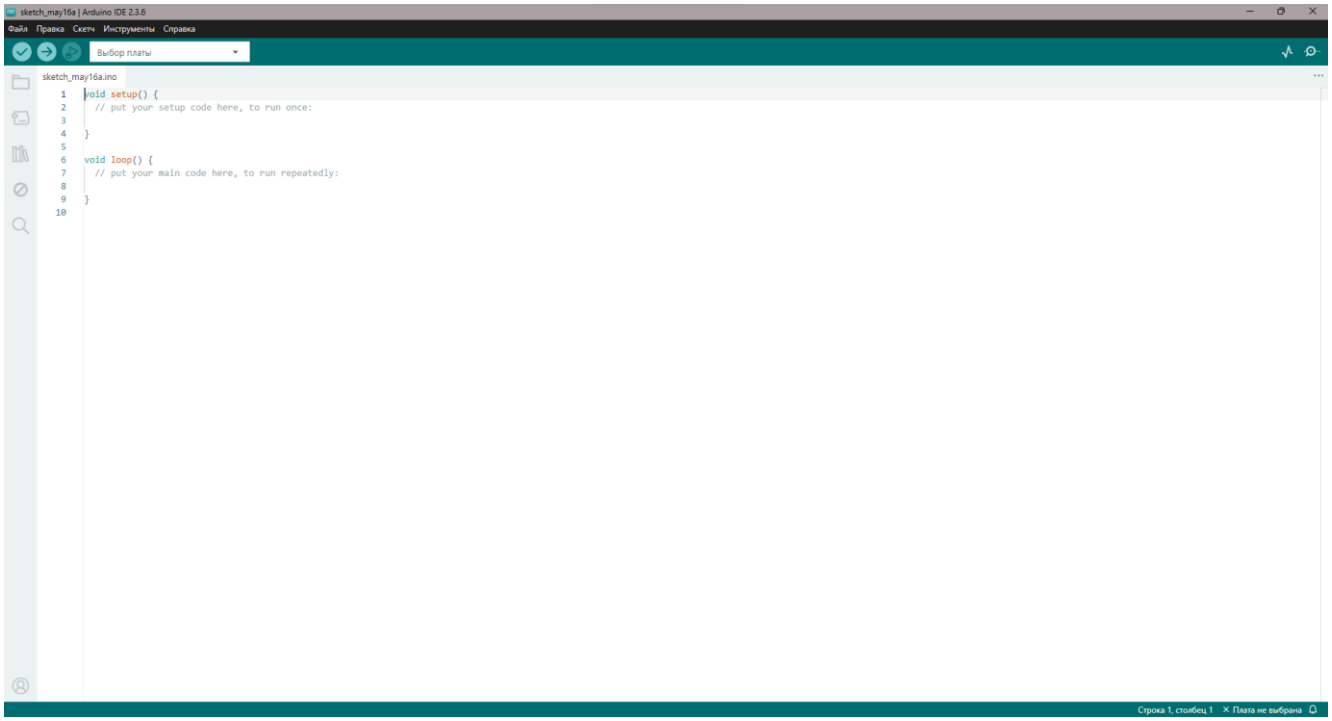


Рисунок 11 – Програмне середовище Arduino IDE

Інтерфейс Arduino IDE порівняно простий у освоєнні, його основою є C++ подібна мова програмування з певними функціями. Для програмування Arduino використається спрощена версія мови C++. Як і в інших Сі-подібних мовах

програмування є ряд правил написання коду. Так само як і C++ мова є жорстко типізованою та компільованою. Також Arduino IDE має широкий спектр підтримуваних плат, таких як Arduino UNO, Arduino Nano, ESP32, ESP8266, сімейство мікроконтролерів Attiny тощо. Тут є можливість підключати майже будь-яку плату за допомогою підтримки плат через вбудований менеджер Arduino IDE.

Після встановлення Arduino IDE необхідно підключити підтримку плати ESP32 через менеджер плат, додавши спеціальну URL до платформи Espressif. Далі потрібно встановити спеціальний драйвер для чипу CH340, щоб комп’ютер зміг ініціалізувати мікроконтролер та розпізнати як пристрій. Наступним кроком потрібно через менеджер бібліотек Arduino потрібні встановити бібліотеки:

– Adafruit BME280 для зчитування кліматичних параметрів з датчика;
– GxEPD & GxEPD2 для роботи з E-Ink дисплеєм;
– WiFi.h для підключення до мережі;
– THHPClient.h для взаємодії з OpenWeatherMap API;
– ArduinoJson для розбору JSON-даних з інтернету.

Далі потрібно перейти в спеціальну вкладку «інструменти» та вибрати плату ESP32 Dev Module і в той же вкладці вибрати відповідний COM порт після підключення плати до комп’ютеру.

Після підключення плати до комп’ютеру та її обирання в програмному середовищі потрібно перевірити, чи все працює вірно за допомогою простого скетча, який виводить текст у Serial Monitor та перевіряє зв’язок, що відображено на рисунку 12.

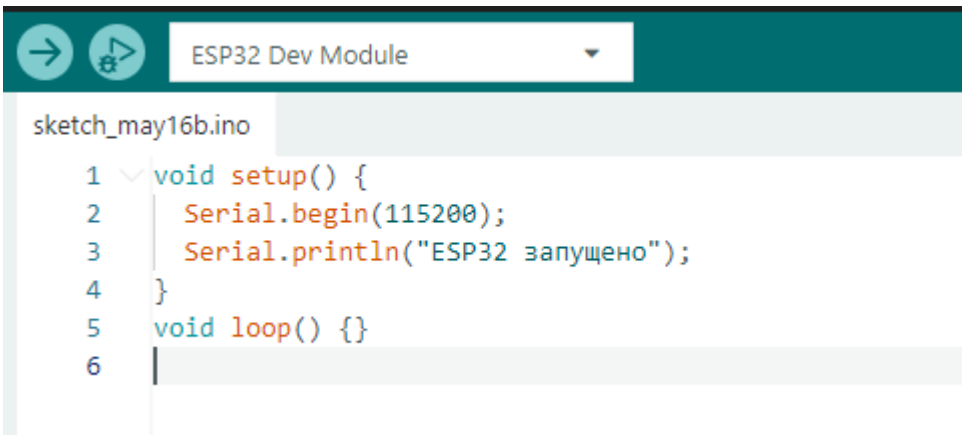


Рисунок 12 – Скетч для перевірки плати

Для роботи датчика BME280 потрібна бібліотека Adafruit BME280, яка призначена для спрощеної роботи з цифровим датчиком BME280, який виробляє компанія Bosch. Бібліотека реалізована на базі Arduino-платформи та оптимізована для використання в мікроконтролерах, таких як ESP32.

За допомогою цієї бібліотеки та датчику можливо налаштувати:

– Зчитування температури у градусах Цельсія або Фаренгейта;

– Вимірювання атмосферного тиску в паскаляха(ПА), гПа або мм рт.ст.;

– Підтримку протоколів I2C та SPI для зв’язку з мікроконтролером;

– Отримання значення вологості в процентах;

– Розрахунок висоти над рівнем моря на основі тиску;

– Виявлення помилок з’єднання якщо датчик не з’єднаний.

При використанні даної бібліотеки ми можемо отримати такі переваги та особливості:

– Документовані приклади і стабільна інтеграція з іншими бібліотеками Arduino;

– Повністю готове API для роботи з BME280 без необхідності вручну надсилати I2C або SPI команди;

– Викоса надійність за рахунок активної підтримки спільноти Adafruit.

– Підтримка як апаратного, так і програмного I2C;

– Працює з широким спектром платформ, таких як Esp8266, ESP32, Arduino UNO, Mega тощо;

– Дає можливість використовувати додаткові функції датчика, наприклад як зміна часу вибірки, калібрування, тощо.

А для підключення та управління E-Ink дисплеєм потрібна GxEPD2 бібліотека для роботи з дисплеями від виробників WeActStudio, Waveshare, Dalian Good Display та їм подібних виробників. Дані дисплеї відомі низьким енергоспоживанням, здатністю зберігати зображення без живлення та високою контрастністю. Дана бібліотека має такі можливості як:

– Вивід графіки у вигляді ліній, прямокутників, кола та тексту і зображень;

– Повну підтримку E-Ink дисплеїв з інтерфейсом SPI;

– Підтримку дисплеїв з різною роздільною здатністю, а саме 1.54”, 2.13”, 2.9”, 4.2”, 7.5”;

– Інтеграцію з Adaftuit GFX для використання знайомого API.

Також дана бібліотека має такі переваги і особливості при використанні з ESP32:

– Часткове оновлення яке дозволяє швидше оновлювати лише частину зображення, що є важливим для зниження енергоспоживання;

– Потужний API для кастомного відображення інформації температури, часу, піктограм погоди тощо;

– Сумісність із великою кількістю дисплеїв;

– Добре документовані приклади;

– Потрібно забезпечити стабільне живлення для E-Ink дисплея, так як деякі моделі вимагають 3.3V з струмом до 20-50 мА під час оновлення;

– Після оновлення екрану дисплей може залишатися в режимі «Busy» протягом кількох секунд;

– Можливе використання будь-яких доступних GPIO SPI;

– Дисплей чутливий до температур і при низьких температурах швидкість оновлення падає.

Для роботи з бездротовим з’єднанням потрібна бібліотека WiFi.h яка використовується для керування бездротовим з’єднанням на мікроконтролерах по типу ESP32. Дана бібліотека дозволяє ESP32 підключатися до мереж Wi-Fi,

організовувати клієнт-серверну взаємодію, створювати власну точку доступу, а також підтримує безпечні з’єднання. Дана бібліотека надає такі можливості:

– Підключення до мережі Wi-Fi у режимі клієнта (STA);

– Створення точки доступу з власною мережею;

– Робота у змішаному режимі STA + AP;

– Автоматичне повторне підключення при втраті мережі;

– Отримання інформації про стан з’єднання, а саме об IP-адресі, RSSI, SSID;

– Підтримка DHCP, статичних IP, WPA/WPA2.

І має такі переваги при використанні з ESP32 як:

– Можливість підключення до інтернету;

– Швидке з’єднання завдяки збереженню параметрів попередніх підключень;

– Просте API для інтеграції з HTTP, MQTT, OTA та іншими мережевими протоколами;

– Підтримка енергозберігаючих режимів (можна вимикати Wi-Fi між циклами оновлення даних);

– Висока сумісність із іншими бібліотеками (наприклад, HTTPClient, ArduinoJson).

При використанні в метеостанції дана бібліотека використовується для встановлення з’єднання з інтернетом з метою надсилання запитів до OpenWeatherMap API та завантаження актуального прогнозу погоди на поточний час.

Після успішного з’єднання передаються HTTP-запити через іншу бібліотеку – HTTPClient.h яка дозволяє мікроконтролеру ESP32 надсилати HTTP-запити такі як GET, POST, PUT, DELETE тощо. У проєкті вона використовується для отримання погодних даних із OpenWeatherMap API у форматі JSON через HTTP-запити. Дана бібліотека надає такі можливості як:

Надсилання запитів GET/POST до зовнішніх або локальних API;

– Підтримка заголовків, авторизації, HTTP-статусів;

– Робота з HTTPS (за підтримки TLS-сертифікатів);

– Зчитування текстових і JSON-відповідей від сервера;
– Підтримка переадресацій HTTP 301/302;
– Проста інтеграція з бібліотекою WiFi.h.

В розробленій метеостанції дана бібліотека використовується для надсилання періодичних запитів до OpenWeatherMap API та отримання поточної інформації про вологість, тиск, температуру та погодні умови.

Отримані дані розбираються за допомогою бібліотеки ArduinoJson яка дозволяє ефективно зчитувати, створювати та обробляти дані у форматі JSON на мікроконтролерах, таких як ESP32. У проєкті вона потрібна для розбору отриманих даних від OpenWeatherMap API, яка надходить у вигляді JSON-структури з великою кількістю вкладених елементів.

Після отримання відповіді від OpenWeatherMap API ArduinoJson швидко та зручно дістає лише ті параметри, які необхідні для відображення на екрані, а саме температуру, тиск, вологість та стан погоди.

ВИСНОВКИ

У ході виконаної кваліфікаційної роботи було спроєктовано та реалізовано прототип автономної метеостанції з використанням мікроконтролера ESP32, датчика BME280, енергоощадного E-Ink дисплея на 4.2” та Li-Pol акумулятора. Основною метою проєкту було створити компактний, енергоефективний та функціональний пристрій для вимірювання кліматичних параметрів з можливістю роботи в автономному режимі.

У процесі створення та розробки було детально проаналізовано існуючі рішення, вивчено особливості апаратних компоненті та підібрано оптимальні засоби для реалізації поставленої задачі. Система успішно інтегрує локальні дані з датчика BME280 з погодними даними з мережі через API від OpenWeatherMap, що дозволило підвищити точність та повноту представленої інформації.

Особливу увагу було приділено енергозбереженню, так як завдяки використанню E-Ink дисплею та реалізації режиму глибокого сну мікроконтролера, було досягнуто значного зменшення енергоспоживання, що дозволило пристрою працювати протягом тривалого часу без підзарядки. Також було розроблено прости та зручний користувацький інтерфейс для відображення поточних метеоданих з регулярними оновленнями за допомогою інтервалів які можливо налаштувати відповідно до вимог енергоспоживання.

Програмне забезпечення розроблено на базі екосистеми Arduino з використанням бібліотек Adafruit BME280, GxEPD2, WiFi.h, HTTPClient.h та ArduinoJson, які забезпечать надійну взаємодію між апаратними та мережевими компонентами.

Результати пророблені роботи свідчать про ефективність обраного підходу та підтверджують можливість створення доступного та масштабованого рішення для моніторингу погодним умов у реальному часі. Розроблений пристрій можливо використовувати як у побутових, так і в навчальних, аграрних чи екологічних сферах, що робить його універсальним. У майбутньому можливе розширення

функціональності за допомогою додавання нових датчиків, передачі даних на сервер або інтеграцію з системою розумний дім.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ESP32 | Wikipedia. URL: https://en.wikipedia.org/wiki/ESP32
2. Плата разработчика ESP-WROOM-32 ESP32 30 Pin | Arduinoshop. Url: https://ardushop.in.ua/ru/arduino/developer-board-esp-wroom-32-esp-32-wi-fi-bluetooth
3. Модуль ESP32 WROOM-32 4MB Type-C (30-pin) | Arduino.ua. Url: https://arduino.ua/prod7342-wi-fi-modyl-esp32-wroom-32-4mb-type-c-30-pin
4. IoT-платформа ESP32 DevKit v1 на базі мікроконтролера ESP-WROOM-32 | DevDotNet.ORG. Url: https://devdotnet.org/post/iot-platforma-esp32-devkit-v1-na-baze-mcu-esp-wroom-32/
5. Барометр BME280 5B I2C (датчик температури, вологості, тиску) | Arduino.ua. Url: https://arduino.ua/prod1930-bme280-5v-i2c-datchik-temperatyri-vlajnosti-davleniya?
6. BME280 - датчик атмосферного тиску | Avislab. Url: https://blog.avislab.com/bme280/
7. Електронний папір | Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%BF%D0%B0%D0%BF%D1%96%D1%80
8. Принцип роботи дисплея E-ink) | Arduino.ua. Url: https://arduino.ua/art121-princip-raboti-displeya-e-ink?srsltid=AfmBOorpKgCtK1p5g68ygUtxjUu1aGumpaEUaKF-N74r5fqNb4SZY_0w
9. Літій-полімерний акумулятор | Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D1%82%D1%96%D0%B9-%D0%BF%D0%BE%D0%BB%D1%96%D0%BC%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BA%D1%83%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80

10. Що необхідно знати про літій-іонні акумулятори? | BoschTech. Url: https://bosch-tech.net.ua/uk/blog/9_lithium-batteries-uk
11. Searching for this exact esp32 | FritzingForum. Url: https://forum.fritzing.org/t/searching-for-this-exact-esp32/28706
12. Help Request: I'm trying to connect multiple components more efficiently | FritzingForum. Url: https://forum.fritzing.org/t/help-request-im-trying-to-connect-multiple-components-more-efficiently/9443
13. Paragon Project: ESP32 Weather Display | PiCockpit. Url: https://picockpit.com/raspberry-pi/paragon-project-esp32-weather-display/
14. Software for Arduino. | Arduino. Url: https://www.arduino.cc/en/software/
15. Arduino | Wikipedia. Url: https://uk.wikipedia.org/wiki/Arduino
16. Arduino IDE | IT MASTER. Url: https://itmaster.biz.ua/electronics/arduino/arduino-ide.html
17. Adafruit_BME280_Library | GitHub. Url: https://github.com/adafruit/Adafruit_BME280_Library
18. GxEPD2 | GitHub. Url: https://github.com/ZinggJM/GxEPD2
19. Бібліотека WiFi | Arduino.ua | https://doc.arduino.ua/ru/prog/WiFi
20. ESP32 Deep Sleep with Arduino IDE and Wake Up Sources | Random Nerd Tutorials. URL: randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources
21. ESP32 Timer Wake Up from Deep Sleep | Random Nerd Tutorials. URL: randomnerdtutorials.com/esp32-timer-wake-up-deep-sleep
22. ESP32 External Wake Up from Deep Sleep | Random Nerd Tutorials. URL: randomnerdtutorials.com/esp32-external-wake-up-deep-sleep
23. MicroPython: ESP32 Deep Sleep and Wake Up Sources | Random Nerd Tutorials. URL: randomnerdtutorials.com/micropython-esp32-deep-sleep-wake-up-sources
24. How to Use ESP32 Deep Sleep Mode for Battery-Powered IoT Projects | Best Engineering Projects. URL: bestengineeringprojects.com/how-to-use-esp32-deep-sleep-mode-for-battery-powered-iot-projects
25. ESP32 Deep Sleep Tutorial | Instructables. URL: www.instructables.com/ESP32-

Deep-Sleep-Tutorial

26. ESP32 Deep Sleep Tutorial: Maximize Battery Efficiency | DFRobot Community. URL: community.dfrobot.com/makelog-314135.html
27. GxEPD2 – Arduino Display Library for SPI E-Paper Displays | Arduino Docs. URL: docs.arduino.cc/libraries/gxepd2/
28. GxEPD2 (GitHub) — Arduino library for e-paper displays. URL: github.com/ZinggJM/GxEPD2
29. How to use a 4-Color ePaper with GxEPD2 and XIAO | Hackster.io. URL: www.hackster.io/etolocka/how-to-use-a-4-color-epaper-with-gxepd2-and-xiao-0e95a2
30. E-paper display using partial updates | Arduino Project Hub. URL: projecthub.arduino.cc/galoebn/e-paper-display-using-partial-updates-a401

Додаток А

ПРОГРАМНИЙ КОД

```
#include "owm_credentials.h"  // See 'owm_credentials' tab and enter your OWM API key and set the Wifi SSID and PASSWORD
#include <ArduinoJson.h>       // https://github.com/bblanchon/ArduinoJson
#include <WiFi.h>              // Built-in
#include "time.h"              // Built-in
#include <SPI.h>               // Built-in
#define  ENABLE_GxEPD2_display 0
#include <GxEPD2_BW.h>
#include <GxEPD2_3C.h>
#include <U8g2_for_Adafruit_GFX.h>
#include "epaper_fonts.h"
#include "forecast_record.h"
#include "lang.h"

#define SCREEN_WIDTH  400.0
#define SCREEN_HEIGHT 300.0

enum alignment {LEFT, RIGHT, CENTER};

// Connections for e.g. LOLIN D32
static const uint8_t EPD_BUSY = 4;  // to EPD BUSY
static const uint8_t EPD_CS   = 5;  // to EPD CS
static const uint8_t EPD_RST  = 16; // to EPD RST
static const uint8_t EPD_DC   = 17; // to EPD DC
static const uint8_t EPD_SCK  = 18; // to EPD CLK
static const uint8_t EPD_MISO = 19; // Master-In Slave-Out not used, as no data from display
```

```
static const uint8_t EPD_MOSI = 23; // to EPD DIN

GxEPD2_BW<GxEPD2_420_GDEY042T81,
GxEPD2_420_GDEY042T81::HEIGHT>
display(GxEPD2_420_GDEY042T81(/*CS=D8*/  EPD_CS,  /*DC=D3*/  EPD_DC,
/*RST=D4*/ EPD_RST, /*BUSY=D2*/ EPD_BUSY));
U8G2_FOR_ADAFRUIT_GFX u8g2Fonts;

String version = "12.5";

boolean LargeIcon = true, SmallIcon = false;
#define Large  11          // For icon drawing, needs to be odd number for best effect
#define Small  5           // For icon drawing, needs to be odd number for best effect
String  time_str, date_str; // strings to hold time and received weather data
int     wifi_signal, CurrentHour = 0, CurrentMin = 0, CurrentSec = 0;
long    StartTime = 0;

//################# PROGRAM VARIABLES and OBJECTS ################

#define max_readings 24

Forecast_record_type  WxConditions[1];
Forecast_record_type  WxForecast[max_readings];

#include <common.h>

#define autoscale_on  true
#define autoscale_off false
#define barchart_on   true
#define barchart_off  false
```

```
float pressure_readings[max_readings]    = {0};
float temperature_readings[max_readings] = {0};
float humidity_readings[max_readings]    = {0};
float rain_readings[max_readings]        = {0};
float snow_readings[max_readings]        = {0};

long SleepDuration = 30; // Sleep time in minutes, aligned to the nearest minute boundary, so if 30 will always update at 00 or 30 past the hour
int  WakeupTime    = 7;  // Don't wakeup until after 07:00 to save battery power
int  SleepTime     = 23; // Sleep after (23+1) 00:00 to save battery power

//#########################################################################################
void setup() {
  StartTime = millis();
  Serial.begin(115200);
  if (StartWiFi() == WL_CONNECTED && SetupTime() == true) {
    if (CurrentHour >= WakeupTime && CurrentHour <= SleepTime ) {
      InitialiseDisplay(); // Give screen time to initialise by getting weather data!
      byte Attempts = 1;
      bool RxWeather = false, RxForecast = false;
      WiFiClient client;   // wifi client object
      while ((RxWeather == false || RxForecast == false) && Attempts <= 2) { // Try up-to 2 time for Weather and Forecast data
        if (RxWeather  == false) RxWeather  = obtain_wx_data(client, "weather");
        if (RxForecast == false) RxForecast = obtain_wx_data(client, "forecast");
        Attempts++;
      }
      if (RxWeather && RxForecast) { // Only if received both Weather or Forecast
```

```
proceed
        StopWiFi(); // Reduces power consumption
        DisplayWeather();
        display.display(false); // Full screen update mode
      }
    }
  }
  BeginSleep();
}
//##################################################################################################
void loop() {
}
//##################################################################################################
void BeginSleep() {
  display.powerOff();
  long SleepTimer = SleepDuration * 60; // theoretical sleep duration
  long offset = (CurrentMin % SleepDuration) * 60 + CurrentSec; // number of seconds elapsed after last theoretical wake-up time point
  if (offset > SleepDuration/2 * 60){ // waking up too early will cause <offset> too large
    offset -= SleepDuration * 60; // then we should make it negative, so as to extend this coming sleep duration
  }
  esp_sleep_enable_timer_wakeup((SleepTimer - offset) * 1000000LL); // do compensation to cover ESP32 RTC timer source inaccuracies
#ifdef BUILTIN_LED
  pinMode(BUILTIN_LED, INPUT); // If it's On, turn it off and some boards use GPIO-5 for SPI-SS, which remains low after screen use
  digitalWrite(BUILTIN_LED, HIGH);
```

```
#endif
  Serial.println("Entering " + String(SleepTimer) + "-secs of sleep time");
  Serial.println("Awake for : " + String((millis() - StartTime) / 1000.0, 3) + "-secs");
  Serial.println("Starting deep-sleep period...");
  esp_deep_sleep_start();      // Sleep for e.g. 30 minutes
}
//#########################################################################
#####################
void DisplayWeather() {               // 4.2" e-paper display is 400x300 resolution
  DrawHeadingSection();               // Top line of the display
  DrawMainWeatherSection(172, 70);         // Centre section of display for Location,
temperature, Weather report, current Wx Symbol and wind direction
  DrawForecastSection(233, 15);        // 3hr forecast boxes
  DisplayPrecipitationSection(233, 82); // Precipitation sectio
  if        (WxConditions[0].Visibility        >        0)        Visibility(335,        100,
String(WxConditions[0].Visibility) + "M");
  if        (WxConditions[0].Cloudcover        >        0)        CloudCover(350,        125,
WxConditions[0].Cloudcover);
  DrawAstronomySection(233, 74);        // Astronomy section Sun rise/set, Moon phase
and Moon icon
}
//#########################################################################
#####################
void DrawHeadingSection() {
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  drawString(SCREEN_WIDTH / 2, 0, City, CENTER);
  drawString(SCREEN_WIDTH, 0, date_str, RIGHT);
  drawString(4, 0, time_str, LEFT);
  DrawBattery(65, 12);
  display.drawLine(0, 12, SCREEN_WIDTH, 12, GxEPD_BLACK);
```

```
}
//#########################################################################
#####################
void DrawMainWeatherSection(int x, int y) {
  DisplayDisplayWindSection(x - 115, y - 3, WxConditions[0].Winddir,
WxConditions[0].Windspeed, 40);
  DisplayWXicon(x + 5, y - 5, WxConditions[0].Icon, LargeIcon);
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);
  DrawPressureAndTrend(x - 120, y + 58, WxConditions[0].Pressure,
WxConditions[0].Trend);
  u8g2Fonts.setFont(u8g2_font_helvB12_tf);
  String Wx_Description = WxConditions[0].Forecast0;
  if (WxConditions[0].Forecast1 != "") Wx_Description += " & " +
WxConditions[0].Forecast1;
  if (WxConditions[0].Forecast2 != "" && WxConditions[0].Forecast1 !=
WxConditions[0].Forecast2) Wx_Description += " & " +  WxConditions[0].Forecast2;
  drawStringMaxWidth(x - 170, y + 83, 28, TitleCase(Wx_Description), LEFT);
  DrawMainWx(x, y + 60);
  display.drawRect(0, y + 68, 232, 48, GxEPD_BLACK);
}
//#########################################################################
#####################
void DrawForecastSection(int x, int y) {
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);
  DrawForecastWeather(x, y, 0);
  DrawForecastWeather(x + 56, y, 1);
  DrawForecastWeather(x + 112, y, 2);
  //     (x,y,width,height,MinValue, MaxValue, Title, Data Array, AutoScale, ChartMode)
  for (int r = 0; r < max_readings; r++) {
    if (Units == "I") {
```

```
      pressure_readings[r] = WxForecast[r].Pressure * 0.02953;
      rain_readings[r]     = WxForecast[r].Rainfall * 0.0393701;
    }
    else {
      pressure_readings[r] = WxForecast[r].Pressure;
      rain_readings[r]     = WxForecast[r].Rainfall;
    }
    temperature_readings[r] = WxForecast[r].Temperature;
  }
  display.drawLine(0, y + 172, SCREEN_WIDTH, y + 172, GxEPD_BLACK);
  u8g2Fonts.setFont(u8g2_font_helvB12_tf);
  drawString(SCREEN_WIDTH / 2, y + 180, TXT_FORECAST_VALUES, CENTER);
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);
  DrawGraph(SCREEN_WIDTH / 400 * 30,  SCREEN_HEIGHT / 300 * 221, SCREEN_WIDTH / 4, SCREEN_HEIGHT / 5, 900, 1050, Units == "M" ? TXT_PRESSURE_HPA : TXT_PRESSURE_IN, pressure_readings, max_readings, autoscale_on, barchart_off);
  DrawGraph(SCREEN_WIDTH / 400 * 158, SCREEN_HEIGHT / 300 * 221, SCREEN_WIDTH / 4, SCREEN_HEIGHT / 5, 10, 30, Units == "M" ? TXT_TEMPERATURE_C : TXT_TEMPERATURE_F, temperature_readings, max_readings, autoscale_on, barchart_off);
  DrawGraph(SCREEN_WIDTH / 400 * 288, SCREEN_HEIGHT / 300 * 221, SCREEN_WIDTH / 4, SCREEN_HEIGHT / 5, 0, 30, Units == "M" ? TXT_RAINFALL_MM : TXT_RAINFALL_IN, rain_readings, max_readings, autoscale_on, barchart_on);
}
//#########################################################################################
void DrawForecastWeather(int x, int y, int index) {
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
```

```
  display.drawRect(x, y, 55, 65, GxEPD_BLACK);
  display.drawLine(x + 1, y + 13, x + 54, y + 13, GxEPD_BLACK);
  DisplayWXicon(x + 28, y + 35, WxForecast[index].Icon, SmallIcon);
  drawString(x + 31, y + 3, String(ConvertUnixTime(WxForecast[index].Dt + WxConditions[0].Timezone).substring(0,5)), CENTER);
  drawString(x + 41, y + 52, String(WxForecast[index].High, 0) + "° / " + String(WxForecast[index].Low, 0) + "°", CENTER);
}
//#########################################################################################
void DrawMainWx(int x, int y) {
  u8g2Fonts.setFont(u8g2_font_helvB14_tf);
  drawString(x - 25, y - 22, String(WxConditions[0].Temperature, 1) + "°" + (Units == "M" ? "C" : "F"), CENTER); // Show current Temperature
  u8g2Fonts.setFont(u8g2_font_helvB12_tf);
  drawString(x - 15, y - 3, String(WxConditions[0].High, 0) + "° | " + String(WxConditions[0].Low, 0) + "°", CENTER); // Show forecast high and Low
  drawString(x + 30, y - 22, String(WxConditions[0].Humidity, 0) + "%", CENTER);
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);
  drawString(x + 32, y - 3, "RH", CENTER);
}
//#########################################################################################
void DisplayDisplayWindSection(int x, int y, float angle, float windspeed, int Cradius) {
  arrow(x, y, Cradius - 7, angle, 12, 18); // Show wind direction on outer circle of width and length
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  int dxo, dyo, dxi, dyi;
  display.drawLine(0, 15, 0, y + Cradius + 30, GxEPD_BLACK);
  display.drawCircle(x, y, Cradius, GxEPD_BLACK);     // Draw compass circle
```

```
  display.drawCircle(x, y, Cradius + 1, GxEPD_BLACK); // Draw compass circle
  display.drawCircle(x, y, Cradius * 0.7, GxEPD_BLACK); // Draw compass inner circle
  for (float a = 0; a < 360; a = a + 22.5) {
    dxo = Cradius * cos((a - 90) * PI / 180);
    dyo = Cradius * sin((a - 90) * PI / 180);
    if (a == 45)  drawString(dxo + x + 10, dyo + y - 10, TXT_NE, CENTER);
    if (a == 135) drawString(dxo + x + 7,  dyo + y + 5,  TXT_SE, CENTER);
    if (a == 225) drawString(dxo + x - 15, dyo + y,      TXT_SW, CENTER);
    if (a == 315) drawString(dxo + x - 15, dyo + y - 10, TXT_NW, CENTER);
    dxi = dxo * 0.9;
    dyi = dyo * 0.9;
    display.drawLine(dxo + x, dyo + y, dxi + x, dyi + y, GxEPD_BLACK);
    dxo = dxo * 0.7;
    dyo = dyo * 0.7;
    dxi = dxo * 0.9;
    dyi = dyo * 0.9;
    display.drawLine(dxo + x, dyo + y, dxi + x, dyi + y, GxEPD_BLACK);
  }
  drawString(x, y - Cradius - 10,     TXT_N, CENTER);
  drawString(x, y + Cradius + 5,      TXT_S, CENTER);
  drawString(x - Cradius - 10, y - 3, TXT_W, CENTER);
  drawString(x + Cradius + 8,  y - 3, TXT_E, CENTER);
  drawString(x - 2, y - 20, WindDegToDirection(angle), CENTER);
  drawString(x + 3, y + 12, String(angle, 0) + "°", CENTER);
  drawString(x + 3, y - 3, String(windspeed, 1) + (Units == "M" ? "m/s" : "mph"),
CENTER);
}
//#########################################################################
####################
String WindDegToDirection(float winddirection) {
```

```
  int dir = int((winddirection / 22.5) + 0.5);
  String Ord_direction[16] = {TXT_N, TXT_NNE, TXT_NE, TXT_ENE, TXT_E, TXT_ESE, TXT_SE, TXT_SSE, TXT_S, TXT_SSW, TXT_SW, TXT_WSW, TXT_W, TXT_WNW, TXT_NW, TXT_NNW};
  return Ord_direction[(dir % 16)];
}
//#########################################################################################
void DrawPressureAndTrend(int x, int y, float pressure, String slope) {
  drawString(x, y, String(pressure, (Units == "M" ? 0 : 1)) + (Units == "M" ? "mb" : "in"), CENTER);
  x = x + 40; y = y + 2;
  if      (slope == "+") {
    display.drawLine(x,  y, x + 4, y - 4, GxEPD_BLACK);
    display.drawLine(x + 4, y - 4, x + 8, y, GxEPD_BLACK);
  }
  else if (slope == "0") {
    display.drawLine(x + 4, y - 4, x + 8, y, GxEPD_BLACK);
    display.drawLine(x + 4, y + 4, x + 8, y, GxEPD_BLACK);
  }
  else if (slope == "-") {
    display.drawLine(x,  y, x + 4, y + 4, GxEPD_BLACK);
    display.drawLine(x + 4, y + 4, x + 8, y, GxEPD_BLACK);
  }
}
//#########################################################################################
void DisplayPrecipitationSection(int x, int y) {
  display.drawRect(x, y - 1, 167, 56, GxEPD_BLACK); // precipitation outline
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);
```

```
  if (WxForecast[1].Rainfall > 0.005) { // Ignore small amounts
    drawString(x + 5, y + 15, String(WxForecast[1].Rainfall, 2) + (Units == "M" ? "mm" : "in"), LEFT); // Only display rainfall total today if > 0
    addraindrop(x + 65 - (Units == "I" ? 10 : 0), y + 16, 7);
  }
  if (WxForecast[1].Snowfall > 0.005)  // Ignore small amounts
    drawString(x + 5, y + 35, String(WxForecast[1].Snowfall, 2) + (Units == "M" ? "mm" : "in") + " * *", LEFT); // Only display snowfall total today if > 0
}
//#########################################################################################
void DrawAstronomySection(int x, int y) {
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  display.drawRect(x, y + 64, 167, 48, GxEPD_BLACK);
  drawString(x + 7, y + 70, ConvertUnixTime(WxConditions[0].Sunrise + WxConditions[0].Timezone).substring(0, (Units == "M" ? 5 : 7)) + " " + TXT_SUNRISE, LEFT);
  drawString(x + 7, y + 85, ConvertUnixTime(WxConditions[0].Sunset + WxConditions[0].Timezone).substring(0, (Units == "M" ? 5 : 7)) + " " + TXT_SUNSET, LEFT);
  time_t now = time(NULL);
  struct tm * now_utc = gmtime(&now);
  const int day_utc   = now_utc->tm_mday;
  const int month_utc = now_utc->tm_mon + 1;
  const int year_utc  = now_utc->tm_year + 1900;
  drawString(x + 7, y + 100, MoonPhase(day_utc, month_utc, year_utc), LEFT);
  DrawMoon(x + 105, y + 50, day_utc, month_utc, year_utc, Hemisphere);
}
//#########################################################################################
```

```
void DrawMoon(int x, int y, int dd, int mm, int yy, String hemisphere) {
  const int diameter = 38;
  double Phase = NormalizedMoonPhase(dd, mm, yy);
  hemisphere.toLowerCase();
  if (hemisphere == "south") Phase = 1 - Phase;
  // Draw dark part of moon
  display.fillCircle(x + diameter - 1, y + diameter, diameter / 2 + 1, GxEPD_BLACK);
  const int number_of_lines = 90;
  for (double Ypos = 0; Ypos <= 45; Ypos++) {
    double Xpos = sqrt(45 * 45 - Ypos * Ypos);
    // Determine the edges of the lighted part of the moon
    double Rpos = 2 * Xpos;
    double Xpos1, Xpos2;
    if (Phase < 0.5) {
      Xpos1 = - Xpos;
      Xpos2 = (Rpos - 2 * Phase * Rpos - Xpos);
    }
    else {
      Xpos1 = Xpos;
      Xpos2 = (Xpos - 2 * Phase * Rpos + Rpos);
    }
    // Draw light part of moon
    double pW1x = (Xpos1 + number_of_lines) / number_of_lines * diameter + x;
    double pW1y = (number_of_lines - Ypos)  / number_of_lines * diameter + y;
    double pW2x = (Xpos2 + number_of_lines) / number_of_lines * diameter + x;
    double pW2y = (number_of_lines - Ypos)  / number_of_lines * diameter + y;
    double pW3x = (Xpos1 + number_of_lines) / number_of_lines * diameter + x;
    double pW3y = (Ypos + number_of_lines)  / number_of_lines * diameter + y;
    double pW4x = (Xpos2 + number_of_lines) / number_of_lines * diameter + x;
    double pW4y = (Ypos + number_of_lines)  / number_of_lines * diameter + y;
```

```
    display.drawLine(pW1x, pW1y, pW2x, pW2y, GxEPD_WHITE);
    display.drawLine(pW3x, pW3y, pW4x, pW4y, GxEPD_WHITE);
  }
  display.drawCircle(x + diameter - 1, y + diameter, diameter / 2 + 1, GxEPD_BLACK);
}
//#########################################################################
####################
String MoonPhase(int d, int m, int y) {
  int c, e;
  double jd;
  int b;
  if (m < 3) {
    y--;
    m += 12;
  }
  ++m;
  c   = 365.25 * y;
  e   = 30.6  * m;
  jd  = c + e + d - 694039.09;     /* jd is total days elapsed */
  jd /= 29.53059;                  /* divide by the moon cycle (29.53 days) */
  b   = jd;                       /* int(jd) -> b, take integer part of jd */
  jd -= b;                        /* subtract integer part to leave fractional part of original jd */
  b   = jd * 8 + 0.5;              /* scale fraction from 0-8 and round by adding 0.5 */
  b   = b & 7;                    /* 0 and 8 are the same phase so modulo 8 for 0 */
  Hemisphere.toLowerCase();
  if (Hemisphere == "south") b = 7 - b;
  if (b == 0) return TXT_MOON_NEW;              // New;             0%  illuminated
  if (b == 1) return TXT_MOON_WAXING_CRESCENT;  // Waxing crescent; 25% illuminated
  if (b == 2) return TXT_MOON_FIRST_QUARTER;    // First quarter;   50% illuminated
```

```
  if (b == 3) return TXT_MOON_WAXING_GIBBOUS;    // Waxing gibbous; 75% illuminated
  if (b == 4) return TXT_MOON_FULL;              // Full;           100% illuminated
  if (b == 5) return TXT_MOON_WANING_GIBBOUS;    // Waning gibbous; 75% illuminated
  if (b == 6) return TXT_MOON_THIRD_QUARTER;     // Third quarter;  50% illuminated
  if (b == 7) return TXT_MOON_WANING_CRESCENT;   // Waning crescent; 25% illuminated
  return "";
}
//#########################################################################################
void arrow(int x, int y, int asize, float aangle, int pwidth, int plength) {
  float dx = (asize + 28) * cos((aangle - 90) * PI / 180) + x; // calculate X position
  float dy = (asize + 28) * sin((aangle - 90) * PI / 180) + y; // calculate Y position
  float x1 = 0;          float y1 = plength;
  float x2 = pwidth / 2;  float y2 = pwidth / 2;
  float x3 = -pwidth / 2; float y3 = pwidth / 2;
  float angle = aangle * PI / 180;
  float xx1 = x1 * cos(angle) - y1 * sin(angle) + dx;
  float yy1 = y1 * cos(angle) + x1 * sin(angle) + dy;
  float xx2 = x2 * cos(angle) - y2 * sin(angle) + dx;
  float yy2 = y2 * cos(angle) + x2 * sin(angle) + dy;
  float xx3 = x3 * cos(angle) - y3 * sin(angle) + dx;
  float yy3 = y3 * cos(angle) + x3 * sin(angle) + dy;
  display.fillTriangle(xx1, yy1, xx3, yy3, xx2, yy2, GxEPD_BLACK);
}
//#########################################################################################
```

```
void DisplayWXicon(int x, int y, String IconName, bool IconSize) {
  Serial.println(IconName);
  if      (IconName == "01d" || IconName == "01n")  Sunny(x, y, IconSize, IconName);
  else if (IconName == "02d" || IconName == "02n")  MostlySunny(x, y, IconSize, IconName);
  else if (IconName == "03d" || IconName == "03n")  Cloudy(x, y, IconSize, IconName);
  else if (IconName == "04d" || IconName == "04n")  MostlyCloudy(x, y, IconSize, IconName);
  else if (IconName == "09d" || IconName == "09n")  ChanceRain(x, y, IconSize, IconName);
  else if (IconName == "10d" || IconName == "10n")  Rain(x, y, IconSize, IconName);
  else if (IconName == "11d" || IconName == "11n")  Tstorms(x, y, IconSize, IconName);
  else if (IconName == "13d" || IconName == "13n")  Snow(x, y, IconSize, IconName);
  else if (IconName == "50d")                       Haze(x, y, IconSize, IconName);
  else if (IconName == "50n")                       Fog(x, y, IconSize, IconName);
  else                                  Nodata(x, y, IconSize, IconName);
}
//################################################################################
uint8_t StartWiFi() {
  Serial.print("\r\nConnecting to: "); Serial.println(String(ssid));
  IPAddress dns(8, 8, 8, 8); // Google DNS
  WiFi.disconnect();
  WiFi.mode(WIFI_STA); // switch off AP
  WiFi.begin(ssid, password);
  unsigned long start = millis();
  uint8_t connectionStatus;
  bool AttemptConnection = true;
  while (AttemptConnection) {
    connectionStatus = WiFi.status();
```

```
    if (millis() > start + 15000) { // Wait 15-secs maximum
      AttemptConnection = false;
    }
    if (connectionStatus == WL_CONNECTED || connectionStatus == WL_CONNECT_FAILED) {
      AttemptConnection = false;
    }
    delay(50);
  }
  if (connectionStatus == WL_CONNECTED) {
    wifi_signal = WiFi.RSSI(); // Get Wifi Signal strength now, because the WiFi will be turned off to save power!
    Serial.println("WiFi connected at: " + WiFi.localIP().toString());
  }
  else Serial.println("WiFi connection *** FAILED ***");
  return connectionStatus;
}
//#########################################################################################
void StopWiFi() {
  WiFi.disconnect();
  WiFi.mode(WIFI_OFF);
}
//#########################################################################################
boolean SetupTime() {
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer, "time.nist.gov"); //(gmtOffset_sec, daylightOffset_sec, ntpServer)
  setenv("TZ", Timezone, 1);  //setenv()adds the "TZ" variable to the environment with a value TimeZone, only used if set to 1, 0 means no change
```

```
  tzset(); // Set the TZ environment variable
  delay(100);
  bool TimeStatus = UpdateLocalTime();
  return TimeStatus;
}
//#########################################################################
#####################
boolean UpdateLocalTime() {
  struct tm timeinfo;
  char   time_output[30], day_output[30], update_time[30];
  while (!getLocalTime(&timeinfo, 10000)) { // Wait for 5-sec for time to synchronise
    Serial.println("Failed to obtain time");
    return false;
  }
  CurrentHour = timeinfo.tm_hour;
  CurrentMin  = timeinfo.tm_min;
  CurrentSec  = timeinfo.tm_sec;
  //See http://www.cplusplus.com/reference/ctime/strftime/
  //Serial.println(&timeinfo, "%a %b %d %Y   %H:%M:%S"); // Displays: Saturday, June
24 2017 14:05:49
  if (Units == "M") {
    if ((Language == "CZ") || (Language == "DE") || (Language == "NL") || (Language ==
"PL")) {
      sprintf(day_output, "%s, %02u. %s %04u", weekday_D[timeinfo.tm_wday],
timeinfo.tm_mday, month_M[timeinfo.tm_mon], (timeinfo.tm_year) + 1900); //
day_output >> So., 23. Juni 2019 <<
    }
    else
    {
      sprintf(day_output, "%s %02u-%s-%04u", weekday_D[timeinfo.tm_wday],
```

```
timeinfo.tm_mday, month_M[timeinfo.tm_mon], (timeinfo.tm_year) + 1900);
    }
   strftime(update_time, sizeof(update_time), "%H:%M:%S", &timeinfo);  // Creates: '@
14:05:49'                    and                    change                    from                    30                    to                    8
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
   sprintf(time_output, "%s", update_time);
  }
  else
  {
   strftime(day_output, sizeof(day_output), "%a  %b-%d-%Y", &timeinfo); // Creates
'Sat May-31-2019'
   strftime(update_time, sizeof(update_time), "%r", &timeinfo);          // Creates: '@
02:05:49pm'
   sprintf(time_output, "%s", update_time);
  }
  date_str = day_output;
  time_str = time_output;
  return true;
}
//#########################################################################
####################
// Symbols are drawn on a relative 10x10grid and 1 scale unit = 1 drawing unit
void addcloud(int x, int y, int scale, int linesize) {
  //Draw cloud outer
  display.fillCircle(x - scale * 3, y, scale, GxEPD_BLACK);                  // Left most
circle
  display.fillCircle(x + scale * 3, y, scale, GxEPD_BLACK);                  // Right most
circle
  display.fillCircle(x - scale, y - scale, scale * 1.4, GxEPD_BLACK);        // left middle
upper circle
```

```
  display.fillCircle(x + scale * 1.5, y - scale * 1.3, scale * 1.75, GxEPD_BLACK); // Right middle upper circle
  display.fillRect(x - scale * 3 - 1, y - scale, scale * 6, scale * 2 + 1, GxEPD_BLACK); // Upper and lower lines
  //Clear cloud inner
  display.fillCircle(x - scale * 3, y, scale - linesize, GxEPD_WHITE);          // Clear left most circle
  display.fillCircle(x + scale * 3, y, scale - linesize, GxEPD_WHITE);          // Clear right most circle
  display.fillCircle(x - scale, y - scale, scale * 1.4 - linesize, GxEPD_WHITE); // left middle upper circle
  display.fillCircle(x + scale * 1.5, y - scale * 1.3, scale * 1.75 - linesize, GxEPD_WHITE); // Right middle upper circle
  display.fillRect(x - scale * 3 + 2, y - scale + linesize - 1, scale * 5.9, scale * 2 - linesize * 2 + 2, GxEPD_WHITE); // Upper and lower lines
}
//################################################################################
void addraindrop(int x, int y, int scale) {
  display.fillCircle(x, y, scale / 2, GxEPD_BLACK);
  display.fillTriangle(x - scale / 2, y, x, y - scale * 1.2, x + scale / 2, y , GxEPD_BLACK);
  x = x + scale * 1.6; y = y + scale / 3;
  display.fillCircle(x, y, scale / 2, GxEPD_BLACK);
  display.fillTriangle(x - scale / 2, y, x, y - scale * 1.2, x + scale / 2, y , GxEPD_BLACK);
}
//################################################################################
void addrain(int x, int y, int scale, bool IconSize) {
  if (IconSize == SmallIcon) scale *= 1.34;
  for (int d = 0; d < 4; d++) {
```

```
    addraindrop(x + scale * (7.8 - d * 1.95) - scale * 5.2, y + scale * 2.1 - scale / 6, scale / 1.6);
  }
}
//################################################################################################
void addsnow(int x, int y, int scale, bool IconSize) {
  int dxo, dyo, dxi, dyi;
  for (int flakes = 0; flakes < 5; flakes++) {
    for (int i = 0; i < 360; i = i + 45) {
      dxo = 0.5 * scale * cos((i - 90) * 3.14 / 180); dxi = dxo * 0.1;
      dyo = 0.5 * scale * sin((i - 90) * 3.14 / 180); dyi = dyo * 0.1;
      display.drawLine(dxo + x + flakes * 1.5 * scale - scale * 3, dyo + y + scale * 2, dxi + x + 0 + flakes * 1.5 * scale - scale * 3, dyi + y + scale * 2, GxEPD_BLACK);
    }
  }
}
//################################################################################################
void addtstorm(int x, int y, int scale) {
  y = y + scale / 2;
  for (int i = 0; i < 5; i++) {
    display.drawLine(x - scale * 4 + scale * i * 1.5 + 0, y + scale * 1.5, x - scale * 3.5 + scale * i * 1.5 + 0, y + scale, GxEPD_BLACK);
    if (scale != Small) {
      display.drawLine(x - scale * 4 + scale * i * 1.5 + 1, y + scale * 1.5, x - scale * 3.5 + scale * i * 1.5 + 1, y + scale, GxEPD_BLACK);
      display.drawLine(x - scale * 4 + scale * i * 1.5 + 2, y + scale * 1.5, x - scale * 3.5 + scale * i * 1.5 + 2, y + scale, GxEPD_BLACK);
    }
```

```
    display.drawLine(x - scale * 4 + scale * i * 1.5, y + scale * 1.5 + 0, x - scale * 3 + scale * i * 1.5 + 0, y + scale * 1.5 + 0, GxEPD_BLACK);
    if (scale != Small) {
      display.drawLine(x - scale * 4 + scale * i * 1.5, y + scale * 1.5 + 1, x - scale * 3 + scale * i * 1.5 + 0, y + scale * 1.5 + 1, GxEPD_BLACK);
      display.drawLine(x - scale * 4 + scale * i * 1.5, y + scale * 1.5 + 2, x - scale * 3 + scale * i * 1.5 + 0, y + scale * 1.5 + 2, GxEPD_BLACK);
    }
    display.drawLine(x - scale * 3.5 + scale * i * 1.4 + 0, y + scale * 2.5, x - scale * 3 + scale * i * 1.5 + 0, y + scale * 1.5, GxEPD_BLACK);
    if (scale != Small) {
      display.drawLine(x - scale * 3.5 + scale * i * 1.4 + 1, y + scale * 2.5, x - scale * 3 + scale * i * 1.5 + 1, y + scale * 1.5, GxEPD_BLACK);
      display.drawLine(x - scale * 3.5 + scale * i * 1.4 + 2, y + scale * 2.5, x - scale * 3 + scale * i * 1.5 + 2, y + scale * 1.5, GxEPD_BLACK);
    }
  }
}
//################################################################################
void addsun(int x, int y, int scale, bool IconSize) {
  int linesize = 3;
  if (IconSize == SmallIcon) linesize = 1;
  display.fillRect(x - scale * 2, y, scale * 4, linesize, GxEPD_BLACK);
  display.fillRect(x, y - scale * 2, linesize, scale * 4, GxEPD_BLACK);
  display.drawLine(x - scale * 1.3, y - scale * 1.3, x + scale * 1.3, y + scale * 1.3, GxEPD_BLACK);
  display.drawLine(x - scale * 1.3, y + scale * 1.3, x + scale * 1.3, y - scale * 1.3, GxEPD_BLACK);
  if (IconSize == LargeIcon) {
```

```
  display.drawLine(1 + x - scale * 1.3, y - scale * 1.3, 1 + x + scale * 1.3, y + scale * 1.3, GxEPD_BLACK);
  display.drawLine(2 + x - scale * 1.3, y - scale * 1.3, 2 + x + scale * 1.3, y + scale * 1.3, GxEPD_BLACK);
  display.drawLine(3 + x - scale * 1.3, y - scale * 1.3, 3 + x + scale * 1.3, y + scale * 1.3, GxEPD_BLACK);
  display.drawLine(1 + x - scale * 1.3, y + scale * 1.3, 1 + x + scale * 1.3, y - scale * 1.3, GxEPD_BLACK);
  display.drawLine(2 + x - scale * 1.3, y + scale * 1.3, 2 + x + scale * 1.3, y - scale * 1.3, GxEPD_BLACK);
  display.drawLine(3 + x - scale * 1.3, y + scale * 1.3, 3 + x + scale * 1.3, y - scale * 1.3, GxEPD_BLACK);
 }
 display.fillCircle(x, y, scale * 1.3, GxEPD_WHITE);
 display.fillCircle(x, y, scale, GxEPD_BLACK);
 display.fillCircle(x, y, scale - linesize, GxEPD_WHITE);
}
//#########################################################################################
void addfog(int x, int y, int scale, int linesize, bool IconSize) {
 if (IconSize == SmallIcon) {
  y -= 10;
  linesize = 1;
 }
 for (int i = 0; i < 6; i++) {
  display.fillRect(x - scale * 3, y + scale * 1.5, scale * 6, linesize, GxEPD_BLACK);
  display.fillRect(x - scale * 3, y + scale * 2.0, scale * 6, linesize, GxEPD_BLACK);
  display.fillRect(x - scale * 3, y + scale * 2.5, scale * 6, linesize, GxEPD_BLACK);
 }
}
```

```
//#########################################################################
####################
void Sunny(int x, int y, bool IconSize, String IconName) {
  int scale = Small, offset = 3;
  if (IconSize == LargeIcon) {
    scale = Large;
    y = y - 8;
    offset = 18;
  } else y = y - 3; // Shift up small sun icon
  if (IconName.endsWith("n")) addmoon(x, y + offset, scale, IconSize);
  scale = scale * 1.6;
  addsun(x, y, scale, IconSize);
}
//#########################################################################
####################
void MostlySunny(int x, int y, bool IconSize, String IconName) {
  int scale = Small, linesize = 3, offset = 3;
  if (IconSize == LargeIcon) {
    scale = Large;
    offset = 10;
  } else linesize = 1;
  if (IconName.endsWith("n")) addmoon(x, y + offset, scale, IconSize);
  addcloud(x, y + offset, scale, linesize);
  addsun(x - scale * 1.8, y - scale * 1.8 + offset, scale, IconSize);
}
//#########################################################################
####################
void MostlyCloudy(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
```

```
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addsun(x - scale * 1.8, y - scale * 1.8, scale, IconSize);
  addcloud(x, y, scale, linesize);
}
//################################################################################
####################
void Cloudy(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
    scale = Small;
    if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
    linesize = 1;
    addcloud(x, y, scale, linesize);
  }
  else {
    y += 10;
    if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
    addcloud(x + 30, y - 35, 5, linesize); // Cloud top right
    addcloud(x - 20, y - 25, 7, linesize); // Cloud top left
    addcloud(x, y, scale, linesize);       // Main cloud
  }
}
//################################################################################
####################
void Rain(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
```

```
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y + 10, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addrain(x, y, scale, IconSize);
}
//#########################################################################
####################
void ExpectRain(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
  addsun(x - scale * 1.8, y - scale * 1.8, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addrain(x, y, scale, IconSize);
}
//#########################################################################
####################
void ChanceRain(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
```

```
  addsun(x - scale * 1.8, y - scale * 1.8, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addrain(x, y, scale, IconSize);
}
//################################################################################
####################
void Tstorms(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addtstorm(x, y, scale);
}
//################################################################################
####################
void Snow(int x, int y, bool IconSize, String IconName) {
  int scale = Large, linesize = 3;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y + 15, scale, IconSize);
  addcloud(x, y, scale, linesize);
  addsnow(x, y, scale, IconSize);
}
//################################################################################
####################
```

```
void Fog(int x, int y, bool IconSize, String IconName) {
  int linesize = 3, scale = Large;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
  addcloud(x, y - 5, scale, linesize);
  addfog(x, y - 5, scale, linesize, IconSize);
}
//#########################################################################
####################
void Haze(int x, int y, bool IconSize, String IconName) {
  int linesize = 3, scale = Large;
  if (IconSize == SmallIcon) {
    scale = Small;
    linesize = 1;
  }
  if (IconName.endsWith("n")) addmoon(x, y, scale, IconSize);
  addsun(x, y - 5, scale * 1.4, IconSize);
  addfog(x, y - 5, scale * 1.4, linesize, IconSize);
}
//#########################################################################
####################
void CloudCover(int x, int y, int CCover) {
  addcloud(x - 9, y - 3, Small * 0.5, 2); // Cloud top left
  addcloud(x + 3, y - 3, Small * 0.5, 2); // Cloud top right
  addcloud(x, y,         Small * 0.5, 2); // Main cloud
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  drawString(x + 15, y - 5, String(CCover) + "%", LEFT);
```

```
}
//#########################################################################
####################
void Visibility(int x, int y, String Visi) {
  y = y - 3; //
  float start_angle = 0.52, end_angle = 2.61;
  int r = 10;
  for (float i = start_angle; i < end_angle; i = i + 0.05) {
    display.drawPixel(x + r * cos(i), y - r / 2 + r * sin(i), GxEPD_BLACK);
    display.drawPixel(x + r * cos(i), 1 + y - r / 2 + r * sin(i), GxEPD_BLACK);
  }
  start_angle = 3.61; end_angle = 5.78;
  for (float i = start_angle; i < end_angle; i = i + 0.05) {
    display.drawPixel(x + r * cos(i), y + r / 2 + r * sin(i), GxEPD_BLACK);
    display.drawPixel(x + r * cos(i), 1 + y + r / 2 + r * sin(i), GxEPD_BLACK);
  }
  display.fillCircle(x, y, r / 4, GxEPD_BLACK);
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  drawString(x + 12, y - 3, Visi, LEFT);
}
//#########################################################################
####################
void addmoon(int x, int y, int scale, bool IconSize) {
  if (IconSize == LargeIcon) {
    x = x + 12; y = y + 12;
    display.fillCircle(x - 50, y - 55, scale, GxEPD_BLACK);
    display.fillCircle(x - 35, y - 55, scale * 1.6, GxEPD_WHITE);
  }
  else
  {
```

```
    display.fillCircle(x - 20, y - 12, scale, GxEPD_BLACK);
    display.fillCircle(x - 15, y - 12, scale * 1.6, GxEPD_WHITE);
  }
}
//#########################################################################
####################
void Nodata(int x, int y, bool IconSize, String IconName) {
  if (IconSize == LargeIcon) u8g2Fonts.setFont(u8g2_font_helvB24_tf); else
u8g2Fonts.setFont(u8g2_font_helvB10_tf);
  drawString(x - 3, y - 8, "?", CENTER);
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
}
//#########################################################################
####################
void DrawBattery(int x, int y) {
  uint8_t percentage = 100;
  float voltage = analogRead(35) / 4096.0 * 7.46;
  if (voltage > 1 ) { // Only display if there is a valid reading
    Serial.println("Voltage = " + String(voltage));
    percentage = 2836.9625 * pow(voltage, 4) - 43987.4889 * pow(voltage, 3) +
255233.8134 * pow(voltage, 2) - 656689.7123 * voltage + 632041.7303;
    if (voltage >= 4.20) percentage = 100;
    if (voltage <= 3.50) percentage = 0;
    display.drawRect(x + 15, y - 12, 19, 10, GxEPD_BLACK);
    display.fillRect(x + 34, y - 10, 2, 5, GxEPD_BLACK);
    display.fillRect(x + 17, y - 10, 15 * percentage / 100.0, 6, GxEPD_BLACK);
    drawString(x + 65, y - 11, String(percentage) + "%", RIGHT);
    //drawString(x + 13, y + 5,  String(voltage, 2) + "v", CENTER);
  }
}
```

```
//#########################################################################
####################
/* (C) D L BIRD
   This function will draw a graph on a ePaper/TFT/LCD display using data from an array containing data to be graphed.
   The variable 'max_readings' determines the maximum number of data elements for each array. Call it with the following parametric data:
   x_pos - the x axis top-left position of the graph
   y_pos - the y-axis top-left position of the graph, e.g. 100, 200 would draw the graph 100 pixels along and 200 pixels down from the top-left of the screen
   width - the width of the graph in pixels
   height - height of the graph in pixels
   Y1_Max - sets the scale of plotted data, for example 5000 would scale all data to a Y-axis of 5000 maximum
   data_array1 is parsed by value, externally they can be called anything else, e.g. within the routine it is called data_array1, but externally could be temperature_readings
   auto_scale - a logical value (TRUE or FALSE) that switches the Y-axis autoscale On or Off
   barchart_on - a logical value (TRUE or FALSE) that switches the drawing mode between bar and line graphs
   barchart_colour - a sets the title and graph plotting colour
   If called with Y!_Max value of 500 and the data never goes above 500, then autoscale will retain a 0-500 Y scale, if on, the scale increases/decreases to match the data.
   auto_scale_margin, e.g. if set to 1000 then autoscale increments the scale by 1000 steps.
*/
void DrawGraph(int x_pos, int y_pos, int gwidth, int gheight, float Y1Min, float Y1Max, String title, float DataArray[], int readings, boolean auto_scale, boolean barchart_mode)
{
#define auto_scale_margin 0 // Sets the autoscale increment, so axis steps up in units of
```

e.g. 3

```
#define y_minor_axis 5      // 5 y-axis division markers
  float maxYscale = -10000;
  float minYscale =  10000;
  int last_x, last_y;
  float x1, y1, x2, y2;
  if (auto_scale == true) {
    for (int i = 1; i < readings; i++ ) {
      if (DataArray[i] >= maxYscale) maxYscale = DataArray[i];
      if (DataArray[i] <= minYscale) minYscale = DataArray[i];
    }
    maxYscale = round(maxYscale + auto_scale_margin); // Auto scale the graph and round to the nearest value defined, default was Y1Max
    Y1Max = round(maxYscale + 0.5);
    if (minYscale != 0) minYscale = round(minYscale - auto_scale_margin); // Auto scale the graph and round to the nearest value defined, default was Y1Min
    Y1Min = round(minYscale);
  }
  // Draw the graph
  last_x = x_pos + 1;
  last_y = y_pos + (Y1Max - constrain(DataArray[1], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight;
  display.drawRect(x_pos, y_pos, gwidth + 3, gheight + 2, GxEPD_BLACK);
  u8g2Fonts.setFont(u8g2_font_helvB08_tf);
  drawString(x_pos + gwidth / 2, y_pos - 12, title, CENTER);
  // Draw the graph
  last_x = x_pos;
  last_y = y_pos + (Y1Max - constrain(DataArray[1], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight;
  display.drawRect(x_pos, y_pos, gwidth + 3, gheight + 2, GxEPD_BLACK);
```

```
  drawString(x_pos + gwidth / 2, y_pos - 13, title, CENTER);
  // Draw the data
  for (int gx = 0; gx < readings; gx++) {
    y2 = y_pos + (Y1Max - constrain(DataArray[gx], Y1Min, Y1Max)) / (Y1Max - Y1Min) * gheight + 1;
    if (barchart_mode) {
      x2 = x_pos + gx * (gwidth / readings) + 2;
      display.fillRect(x2, y2, (gwidth / readings) - 2, y_pos + gheight - y2 + 2, GxEPD_BLACK);
    }
    else
    {
      x2 = x_pos + gx * gwidth / (readings - 1) + 1; // max_readings is the global variable that sets the maximum data that can be plotted
      display.drawLine(last_x, last_y, x2, y2, GxEPD_BLACK);
    }
    last_x = x2;
    last_y = y2;
  }
  //Draw the Y-axis scale
#define number_of_dashes 15
  for (int spacing = 0; spacing <= y_minor_axis; spacing++) {
    for (int j = 0; j < number_of_dashes; j++) { // Draw dashed graph grid lines
      if (spacing < y_minor_axis) display.drawFastHLine((x_pos + 3 + j * gwidth / number_of_dashes), y_pos + (gheight * spacing / y_minor_axis), gwidth / (2 * number_of_dashes), GxEPD_BLACK);
    }
    if ((Y1Max - (float)(Y1Max - Y1Min) / y_minor_axis * spacing) < 5 || title == TXT_PRESSURE_IN) {
      drawString(x_pos, y_pos + gheight * spacing / y_minor_axis - 5, String((Y1Max -
```

```
(float)(Y1Max - Y1Min) / y_minor_axis * spacing + 0.01), 1), RIGHT);
    }
    else
    {
      if (Y1Min < 1 && Y1Max < 10)
        drawString(x_pos - 3, y_pos + gheight * spacing / y_minor_axis - 5, String((Y1Max
- (float)(Y1Max - Y1Min) / y_minor_axis * spacing + 0.01), 1), RIGHT);
      else
        drawString(x_pos - 3, y_pos + gheight * spacing / y_minor_axis - 5, String((Y1Max
- (float)(Y1Max - Y1Min) / y_minor_axis * spacing + 0.01), 0), RIGHT);
    }
  }
  for (int i = 0; i <= 2; i++) {
    drawString(15 + x_pos + gwidth / 3 * i, y_pos + gheight + 3, String(i), LEFT);
  }
  drawString(x_pos + gwidth / 2, y_pos + gheight + 10, TXT_DAYS, CENTER);
}
//#########################################################################
####################
void drawString(int x, int y, String text, alignment align) {
  int16_t  x1, y1; //the bounds of x,y and w and h of the variable 'text' in pixels.
  uint16_t w, h;
  display.setTextWrap(false);
  display.getTextBounds(text, x, y, &x1, &y1, &w, &h);
  if (align == RIGHT)  x = x - w;
  if (align == CENTER) x = x - w / 2;
  u8g2Fonts.setCursor(x, y + h);
  u8g2Fonts.print(text);
}
//#########################################################################
```

```
####################
void drawStringMaxWidth(int x, int y, unsigned int text_width, String text, alignment align) {
  int16_t  x1, y1; //the bounds of x,y and w and h of the variable 'text' in pixels.
  uint16_t w, h;
  display.getTextBounds(text, x, y, &x1, &y1, &w, &h);
  if (align == RIGHT)  x = x - w;
  if (align == CENTER) x = x - w / 2;
  u8g2Fonts.setCursor(x, y);
  if (text.length() > text_width * 2) {
    u8g2Fonts.setFont(u8g2_font_helvB10_tf);
    text_width = 42;
    y = y - 3;
  }
  u8g2Fonts.println(text.substring(0, text_width));
  if (text.length() > text_width) {
    u8g2Fonts.setCursor(x, y + h + 15);
    String secondLine = text.substring(text_width);
    secondLine.trim(); // Remove any leading spaces
    u8g2Fonts.println(secondLine);
  }
}
//#########################################################################
####################
void InitialiseDisplay() {
  display.init(115200, true, 2, false);
  // display.init(); for older Waveshare HAT's
  SPI.end();
  SPI.begin(EPD_SCK, EPD_MISO, EPD_MOSI, EPD_CS);
  u8g2Fonts.begin(display); // connect u8g2 procedures to Adafruit GFX
```

```
  u8g2Fonts.setFontMode(1);                 // use u8g2 transparent mode (this is default)
  u8g2Fonts.setFontDirection(0);            // left to right (this is default)
  u8g2Fonts.setForegroundColor(GxEPD_BLACK); // apply Adafruit GFX color
  u8g2Fonts.setBackgroundColor(GxEPD_WHITE); // apply Adafruit GFX color
  u8g2Fonts.setFont(u8g2_font_helvB10_tf);  // select u8g2 font from here: https://github.com/olikraus/u8g2/wiki/fntlistall
  display.fillScreen(GxEPD_WHITE);
  display.setFullWindow();
}
```

Додаток Б

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ
БАКАЛАВРА

на тему: «Метеостанція на базі мікроконтролера ESP32»

Виконав студент групи КІД-42
Фещенко Андрій Іванович

Керівник кваліфікаційної роботи:
Антоненко Артем Васильович, К.т.н., доц

Київ – 2025

Мета роботи – Розробка метеостанції на базі мікроконтролера ESP32

Об`єкт дослідження – Проєкт Paragon: ESP32 Weather Display

Предмет дослідження – Метеостанція на базі мікроконтролера ESP32

Актуальність обраної теми

Щоб зрозуміти, чи актуальна ця тема, потрібно взглянути на сучасні проблеми які нас оточують. Погодне середовище, економічна криза з якою зв'язано підвищення цін на продукти та недостаток часу із-за праці. І завдяки розвитку технологій ці проблеми можливо вирішити за допомогою автономних пристроїв, що здатні автоматично збирати, аналізувати та відображати дані з навколишнього середовища і при більших ресурсах зробити майже повністю автономну ферму для вирощування овочів. Це є одне з найбільш можливих способів використання метеостанції, але її також можливо використовувати в побутових потребах або промислових. Також за допомогою метеостанції можливо зробити умний дім, завдяки якому автономно можливо підтримувати температуру в домі і рівень освітлюванності, що допоможе більш ефективніше використовувати і проводити свій час.





Об’єкт дослідження

Об’єктом дослідження кваліфікаційної роботи є проєкт Люка Марцена під назвою «Paragon: ESP32 Weather Display» котрий демонструє добре зроблену та оптимізовану метеостанцію для базових потреб яка демонструє багато нюансів та як зробити подібний прототип і з яких елементів.





Постановка задачі

Автономна система

Розробити незалежну метеостанцію для довготривалої роботи.

Відображення даних

Зрозумілий інтерфейс на E-Paper дисплеї 4.2"

Енергоефективність

Оптимізувати енергоспоживання для тривалої автономії.

Хмарна передача

Опційна підтримка Wi-Fi для отримання даних з мережі.

Архітектура системи та програмна реалізація

ESP32

Центральний мікроконтролер із Wi-Fi модулем.

BME280

Датчик температури, вологості та атмосферного тиску.

E-Paper 4.2"

Енергозберігаючий дисплей з високою чіткістю.

Живлення

Акумуляторна батарея для автономної роботи.

Arduino IDE

Середовище розробки апаратної частини з підтримкою ESP32.

Бібліотеки

За допомогою котрих можливе підключення датчику BME280 та E-Paper дисплею.

Алгоритми

Використані в апаратній частині які обробляють та фільтрують атмосферні дані.

Інтерфейс

Створений інтерфейс котрий легкий для сприйняття на дисплеї.

Структура проєкту

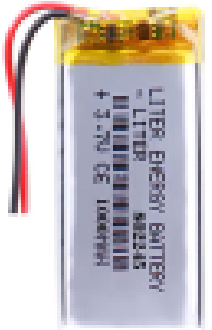
Серцем системи є мікроконтролер ESP32, що забезпечує обчислювальні функції та Wi-Fi-з'єднання. Датчик BME280 відповідає за зчитування температури, вологості та тиску, а інформація виводиться за допомогою дисплею E-Ink діагоналлю 4.2 дюйма. Живлення пристрою забезпечується літій-полімерним акумулятором ємністю 1000 мА·год, що дозволяє системі працювати автономно.

Обґрунтування вибору компонентів





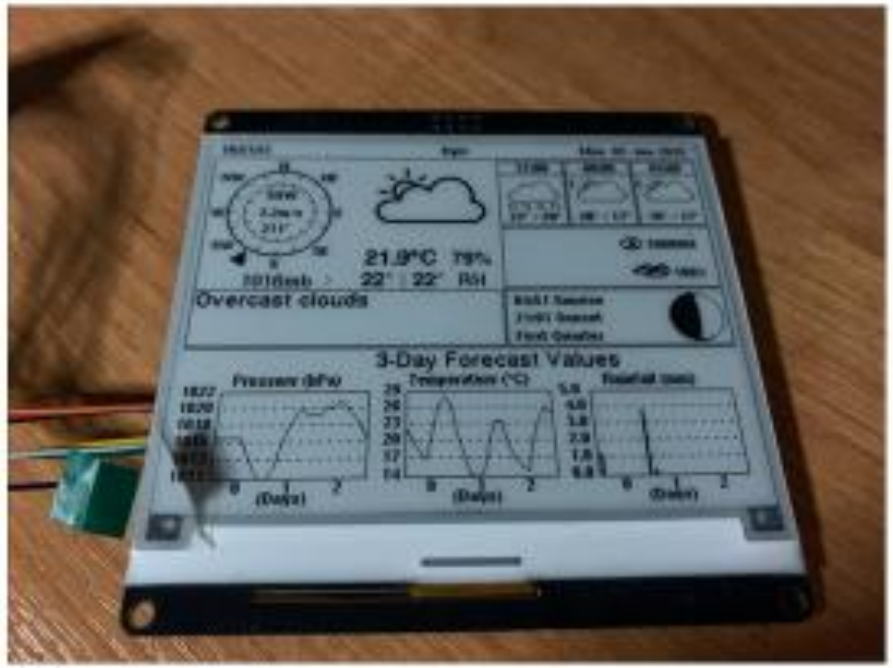




Для проєкту обрано ESP32 завдяки його низькому енергоспоживанню та вбудованому Wi-Fi. BME280 забезпечує надійне зчитування трьох ключових параметрів. Дисплей E-Ink дозволяє знизити споживання енергії, оскільки зображення зберігається навіть при відсутності живлення. Акумулятор 1000 мА·год підтримує тривалу автономну роботу.

Архітектура системи

Мікроконтролер періодично виходить з режиму глибокого сну, зчитує дані з датчика, отримує прогноз погоди з OpenWeatherMap та виводить результати на E-Ink дисплей. Після цього пристрій повертається в енергозберігаючий режим до наступного циклу оновлення.



Програмна частина

Програмна реалізація створена на базі Arduino IDE із застосуванням C++. Для роботи з датчиком, дисплеєм, Wi-Fi та API використовуються спеціалізовані бібліотеки. Система організована таким чином, щоб мінімізувати час активного режиму та оптимізувати використання енергії, за рахунок чого отримаємо автономну станції.



Енергозбереження

Одним з ключових аспектів проєкту є досягнення високого рівня енергоефективності, що дозволяє пристрою працювати автономно тривалий час. Це реалізується завдяки поєднанню апаратних та програмних рішень, орієнтованих на мінімізацію споживання енергії.

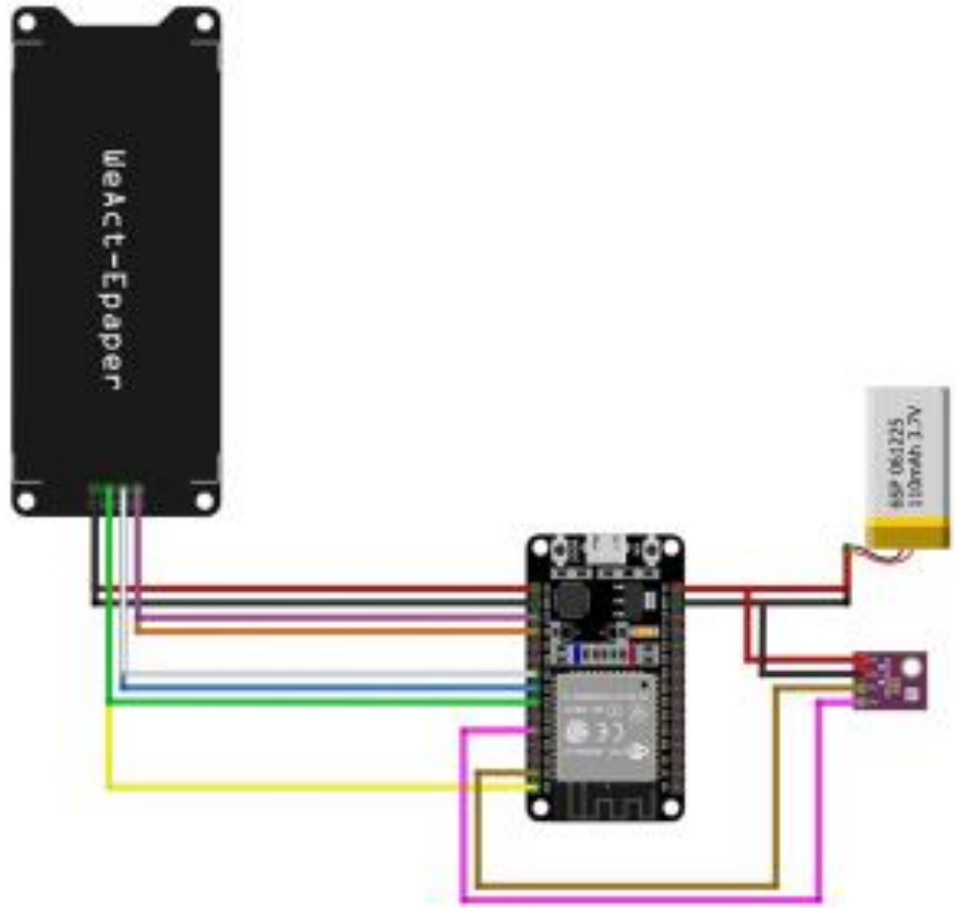
Мікроконтролер ESP32 підтримує режим deep sleep, у якому більшість компонентів вимикаються, а споживання енергії знижується до мікроампер. Пристрій періодично «прокидається» лише для зчитування даних з сенсора, отримання прогнозу з OpenWeatherMap API та оновлення дисплея. Після завершення цих дій ESP32 знову переходить у глибокий сон, що суттєво подовжує автономну роботу.

Крім того, використовується E-Ink дисплей, який не споживає енергію для збереження зображення. Це дозволяє обмежити споживання енергії лише до моменту оновлення інформації, що відбувається один раз на 30 хвилин.

Живлення пристрою забезпечує літій-полімерний акумулятор ємністю 1000 мА·год, який дає змогу працювати в автономному режимі до кількох днів без підзарядки. Завдяки оптимізованій роботі з Wi-Fi (відключення після обміну даними) споживання енергії також зменшується.

Таке поєднання компонентів і програмної логіки робить пристрій придатним для довготривалої роботи в польових умовах без постійного джерела живлення.

Схема підключення



Результати тестування

Метеостанція успішно функціонує в реальних умовах. Пристрій стабільно зчитує показники температури, вологості та тиску, оновлює їх на дисплеї та працює від акумулятора впродовж декількох діб без підзарядки. Отримані результати підтверджують ефективність запропонованого рішення.

ВИСНОВКИ

У ході виконаної дипломної роботи було спроєктовано та реалізовано прототип автономної метеостанції з використанням мікроконтролера ESP32, датчика BME280, енергоощадного E-Ink дисплея на 4.2” та Li-Pol акумулятора. Основною метою проєкту було створити компактний, енергоефективний та функціональний пристрій для вимірювання кліматичних параметрів з можливістю роботи в автономному режимі.

У процесі створення та розробки було детально проаналізовано існуючі рішення, вивчено особливості апаратних компонентів та підібрано оптимальні засоби для реалізації поставленої задачі. Система успішно інтегрує локальні дані з датчика BME280 з погодними даними з мережі через API від OpenWeatherMap, що дозволило підвищити точність та повноту представленої інформації.

Особливу увагу було приділено енергозбереженню, так як завдяки використанню E-Ink дисплею та реалізації режиму глибокого сну мікроконтролера, було досягнуто значного зменшення енергоспоживання, що дозволило пристрою працювати протягом тривалого часу без підзарядки. Також було розроблено прости та зручний користувацький інтерфейс для відображення поточних метеоданих з регулярними оновленнями за допомогою інтервалів які можливо налаштувати відповідно до вимог енергоспоживання.

ПРОДОВЖЕННЯ ВИСНОВКІВ

Програмне забезпечення розроблено на базі екосистеми Arduino з використанням бібліотек Adafruit BME280, GxEPD2, WiFi.h, HTTPClient.h та ArduinoJson, які забезпечать надійну взаємодію між апаратними та мережевими компонентами.

Результати пророблені роботи свідчать про ефективність обраного підходу та підтверджують можливість створення доступного та масштабованого рішення для моніторингу погодним умов у реальному часі. Розроблений пристрій можливо використовувати як у побутових, так і в навчальних, аграрних чи екологічних сферах, що робить його універсальним. У майбутньому можливе розширення функціональності за допомогою додавання нових датчиків, передачі даних на сервер або інтеграцію з системою розумний дім