МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"До захисту допущений"
Зав. кафедри комп'ютерних наук та прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Дослідження та впровадження компактного цифрового аналога полароїдного фотоапарата на платформі ESP32

Виконав: Фугель Владислав Валентинович _____
(прізвище, ім’я, по батькові) (підпис)

група ІПЗ-41

Керівник: Герус Володимир Андрійович, старший викладач кафедри

комп'ютерних наук та прикладної математики _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

Зміст

РЕФЕРАТ

Кваліфікаційна робота: 50 с., 18 рисунків, 10 джерел.

Мета роботи: розробити компактний автономний пристрій на базі ESP32-CAM для захоплення зображень та друку її на чековому папері, а також з можливістю локального керування через веб-інтерфейс.

Об'єкт дослідження: мікроконтролери серії ESP32, побітна обробка зображень та застосування алгоритму Ditherng для покращення якості фото. Друкування відправлених повідомлень та зображень на чековому принтері.

Предмет дослідження: інтеграція ESP32-CAM з OLED-дисплеєм, чековим термопринтером та реалізація веб-інтерфейсу для керування.

Проведено дослідження мережевих протоколів, мікроконтролера, інтегрального принтера та інших електронних компонентів а також бібліотек та протоколів зв’язку з ними. Розроблено модель пристрою для захоплення та друкування фото, доступом до функціоналу через веб-інтерфейс, а саме: відстежування зображекння з камери, налаштування даних точки доступу, друкування повідомлеень та фотографій на принтері.

Ключові слова: ЧЕКОВИЙ ПРИНТЕР, МІКРОКОНТРОЛЕРИ, DITHERING, ВЕБ-ІНТЕРФЕЙС, ПОЛАРОЇД.

ВСТУП

Сучасний розвиток мікроконтролерних технологій відкриває широкі можливості для створення інтелектуальних, автономних пристроїв, здатних виконувати складні завдання із захоплення, обробки та виведення інформації в реальному часі. Завдяки доступності недорогих, але потужних модулів, таких як ESP32-CAM, стало можливим поєднувати комп'ютерний зір, бездротові технології зв'язку та взаємодію з периферійними пристроями у компактних і енергоефективних рішеннях.

Метою цієї роботи є розробка багатофункціонального пристрою на базі ESP32-CAM, який реалізує повний цикл — від захоплення зображення до його виводу на термопринтер після попередньої обробки за алгоритмом дізерингу (англ. dithering). Крім того, пристрій забезпечує візуалізацію поточного зображення з камери на OLED-дисплеї та дає змогу керувати процесом зйомки та друку через вбудований веб-інтерфейс, доступний через локальну WiFi-точку доступу.

Для реалізації такого рішення необхідно було поєднати знання з цифрової обробки зображень, мікроконтролерного програмування, роботи з периферією та веб-розробки. Значну увагу було приділено оптимізації алгоритму дізерингу з урахуванням обмежених ресурсів ESP32 та особливостей термодруку. Особливістю пристрою є можливість ручного регулювання експозиції камери за допомогою потенціометра, що дозволяє адаптувати якість зображення до умов освітлення.

Робота над проектом включає в себе вивчення апаратних можливостей ESP32-CAM, дослідження технологій термодруку на мініатюрних чекових принтерах, реалізацію алгоритмів бінаризації та дізерингу, а також створення зручного веб-інтерфейсу для дистанційного керування функціоналом пристрою. Результатом проекту є повністю автономна система, що демонструє можливості побудови практичних і доступних IoT-рішень у сфері комп'ютерного зору та документування.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗНАЧЕННЯ ТА ПОЯСНЕННЯ КОНЦЕПЦІЙ МЕРЕЖЕВИХ РІШЕНЬ В АВТОНОМНИХ IoT-ПРИСТРОЯХ

1.1. Розробка концепції та вибір напрямку руху

Мережеві рішення в контексті автономних IoT-пристроїв, таких як пристрій на базі ESP32-CAM, охоплюють набір технологій, які дозволяють забезпечити бездротовий зв'язок, дистанційне управління та обмін даними між пристроєм і користувачем через локальну мережу. Основна мета таких рішень — забезпечити надійний та безпечний спосіб інтеракції з пристроєм без необхідності сторонньої інфраструктури або підключення до Інтернету.

Однією з ключових концепцій є локальна точка доступу (WiFi Access Point), яка створюється безпосередньо мікроконтролером ESP32. Такий підхід дозволяє перетворити пристрій на автономний вузол управління, до якого можна підключитися безпосередньо зі смартфона або комп'ютера. Ця модель особливо зручна у випадках, коли пристрій має використовуватись у польових умовах або в середовищі, де немає централізованої мережевої інфраструктури.

Ще однією важливою складовою мережевого рішення є вбудований веб-інтерфейс, реалізований через вбудований HTTP-сервер. Такий інтерфейс дає змогу користувачу виконувати базові операції: перегляд живого зображення з камери, зйомка фото, відправлення зображення на друк або введення текстового повідомлення для друку. Вся логіка обробки запитів відбувається безпосередньо на ESP32, що дозволяє уникнути потреби в сторонньому серверному рішенні.

У межах реалізації пристрою було обрано архітектуру «пристрій-клієнт», де ESP32-CAM виступає як сервер, а пристрої користувачів (телефони, ноутбуки) — як клієнти, що надсилають запити на виконання дій. Така архітектура має ряд переваг:

1) Централізація управління: Весь функціонал зосереджено на одному пристрої, що спрощує оновлення прошивки та полегшує обслуговування.
2) Автономність: Пристрій не залежить від зовнішньої інфраструктури, працює без підключення до Інтернету.

3) Простота розгортання: Достатньо живлення — і пристрій готовий до використання без додаткової конфігурації мережі.
4) Безпека доступу: Можливість встановлення пароля на точку доступу дозволяє обмежити коло користувачів.

Централізована модель комунікації, де ESP32-CAM є єдиною точкою доступу та керування, виявилася найефективнішою в контексті поставлених завдань: зменшення складності системи, забезпечення стабільної взаємодії з користувачем та підтримки повного контролю за процесом зйомки, обробки та друку зображення.

Таким чином, мережеве рішення, реалізоване у вигляді автономної точки доступу з вбудованим веб-інтерфейсом, є оптимальним підходом для забезпечення гнучкого, зручного і безпечного управління пристроєм у реальному часі. Такий підхід дозволяє пристрою бути повністю самодостатнім, мобільним та доступним у широкому спектрі прикладних задач, зокрема в польових умовах, логістиці, інсталяційному мистецтві чи експериментальних дослідженнях.

1.2. Формулювання проблеми та методи дослідження

Проблема:

Розробка автономного мікроконтролерного пристрою з камерою, який здатен виконувати зйомку, обробку зображення, друк та одночасно забезпечувати візуальний і мережевий інтерфейси, потребує збалансованого поєднання апаратних і програмних рішень. Особливо важливим є забезпечення стабільної роботи при обмежених ресурсах та низькому енергоспоживанні.

Аналіз існуючих рішень: Проведено огляд сучасних реалізацій ESP32-проектів із підтримкою веб-інтерфейсу, роботи алгоритму dithering, проектів друку на термопринтері та методів та алгоритмів обробки зображень. Також вивчено приклади реалізації прямих точок доступу (Wi-Fi AP), інтеграції з OLED-дисплеями.

Прототипування та ітеративна розробка: Розробка пристрою велась поетапно із використанням підходу «спроба–помилка», з поступовою інтеграцією апаратних модулів: камери, дисплея, потенціометра та принтера.

Прототипи тестувалися в реальних умовах експлуатації для оцінки стабільності, швидкодії та якості зображення.

Тестування користувацького досвіду: Було створено веб-інтерфейс для управління пристроєм через Wi-Fi. Проведено експериментальне тестування на різних пристроях (смартфон, ноутбук) для виявлення помилок у з'єднанні, взаємодії з інтерфейсом та оцінки зручності використання.

1.3. Вимоги до обладнання та вибір компонентів

Для реалізації пристрою було обрано наступні модулі та компоненти на основі критеріїв компактності, енергоспоживання, функціональності та вартісної доцільності.

1. ESP32-CAM (модуль AI-Thinker) Обґрунтування вибору:
Інтегрована камера, Wi-Fi, Достатня кількість портів для підключення всіх зовнішніх модулів,Висока обчислювальна здатність, компактність та зручність. Заявлені характеристики: максимальний струм – 0.3A, необхідна напруга – 3.3-5V.

2. OLED-дисплей 128×64 (на контролері SSD1306) Обґрунтування вибору:

Низьке енергоспоживання, відсутність проміжних станів що імітує реальне зображення на чеці а також просте I2C-підключення Заявлені характеристики: максимальний струм – 0.04A, необхідна напруга – 5V

3. Потенціометр (аналоговий)

Використовується для ручного регулювання експозиції або яскравості зображення перед обробкою

Підключається до аналогового входу ESP32 і дозволяє в режимі реального часу змінювати параметри зйомки.

4. Термопринтер (TTL/RS232, модель EM5822) Обґрунтування вибору: Сумісність, Ширина паперу, Низьке енергоспоживання у режимі очікування, компактні розміри корпусу без лишніх елементів.

Заявлені характеристики: максимальний струм - 2.5A, необхідна напруга – 6-9V.

1.4. Створення теоретичної схеми пристрою

Посилаючись на вміст підрозділу 1.3. «Вимоги до обладнання та вибір компонентів» . було спроектовано наступну модель взаємодії компонентів: (рис.1.1).

Рис. 1.1. Теоретична схема пристрою

Як видно на схемі, присутні 6 активних та реактивних обов’язкових електронних компонентів, а саме:

1) чековий принтер EM5822
2) OLED екран
3) ESP32
4) Камера
5) Потенціометр
6) Кнопка

Кожен з наих вимагає власного логічного підключення а також для деяких з них потрібне окреме живлення.

Посилаючись на той самий пункт 1.3. «Вимоги до обладнання та вибір компонентів», всіх їх можна розділити на категорії базуючись на необхідному живленні:

Напруга складає 6-9 вольт: чековий принтер EM5822

Напруга складає ~5 вольт: OLED екран, ESP32

Напруга складає 3.3 вольта (живиться від логічного сигналу): ESP32, камера, кнопка, потенціометр

Таким чином можна зробити висновок, що є три рівня напруг, які не можна перемішувати. Також не можна забувати про те, що для правильної передачі логічного сигналу потрібно об’єднати контакт землі або ж ground (gnd) на керуючій платі, ESP32 та на компоненті яким відбувається керування.

Взявши вищесказане до уваги можна прийти до висновку, що потрібно реалізувати поступове живлення згасання живлення від найбільш споживаючих, до найменш споживаючих. Приклад такого рішення зображено на Рис. 1.2.

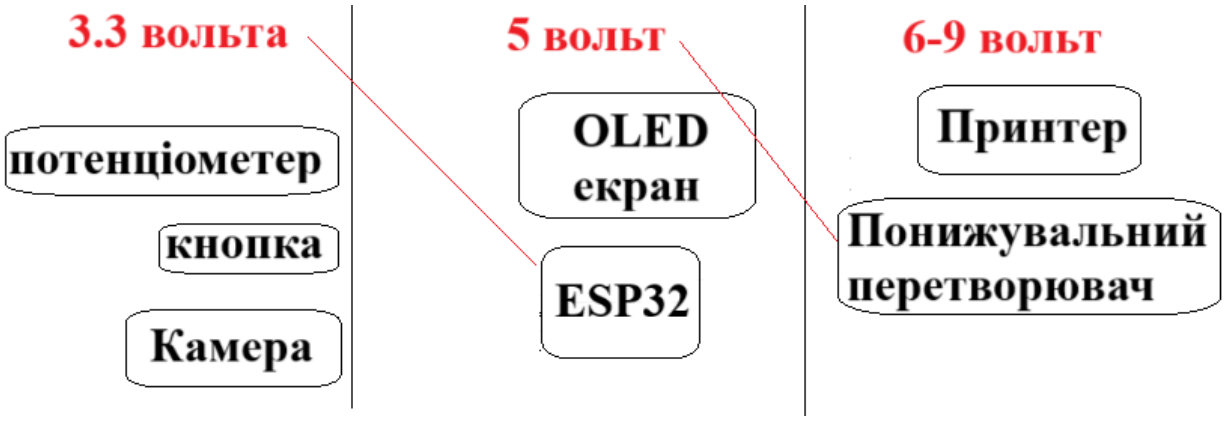

Рис. 1.2. Теоретичний розподіл рівнів напруг пристрою

Таким чином можна прийти до висновку, що потрібно якось отримати джерело живлення напругою від 5 або 6 вольт, та певним струмом який буде рівним сумою струмів всіх активних компонентів, а саме:

чековий принтер EM5822, OLED екран, ESP32, Камера

Звернувшись до пункту пункт 1.3. «Вимоги до обладнання та вибір компонентів» візьмемо струми живлення наведених компонентів та складемо їх:
(чековий принтер EM5822) 2.5A + (OLED екран) 0.0391A +

+ (ESP32) 0.26A + (Камера) 0.04A = ~2.84A

У висновку було визначено, що для адекватної роботи проекту в найбільшій точці навантаження, з врахуванням запасу та падіння напруги, потрібний струм 3 ампера і напруга 6-9 вольт.

РОЗДІЛ 2
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ФОТОАПАРАТІВ ДЛЯ ЧЕКІВ

2.1. Огляд існуючих пристроїв для друку на чеках та фотографування

На ринку з’являється все більше компактних рішень для моментального друку зображень, що поєднують у собі функції цифрового фотоапарата і принтера. Більшість із них орієнтовані на естетику ретро-фотографії, зручність у використанні та миттєве отримання фізичного знімка. Розглянемо найпопулярніші приклади комерційних та DIY-рішень, які використовують чекові принтери або принтери аналогічного типу. Нижче наведено список таких рішень:

1. Instax Mini (Fujifilm) є одним із найвідоміших рішень. Ці пристрої поєднують цифровий фотоапарат із вбудованим мініатюрним принтером, який друкує на спеціальному фотопапері. Хоча вони не використовують класичні термопринтери з чековим папером, сама ідея миттєвого друку фото є ідентичною. Проте ці пристрої залежать від дорогого фотопаперу, що зменшує доступність технології для широкого кола користувачів.

2. Canon Ivy CLIQ / Polaroid Hi-Print реалізували подібну до попереднього концепцію. Вони також використовують фотореактивний термопапір (ZINK), дозволяють зробити знімок і миттєво роздрукувати його. Основним недоліком цих систем є обмеження щодо розміру зображення та висока вартість витратних матеріалів.

3. PeriPage / Poooli / Paperan - це компактні Bluetooth-принтери, які можуть друкувати на термопапері, схожому на чековий. Хоча самі пристрої не мають вбудованих камер, вони широко використовуються в парі зі смартфонами для друку фото, нотаток або QR-кодів. Деякі DIY-проекти інтегрують камери ESP32-CAM із цими принтерами, створюючи дешеві системи для друку простих зображень у чорно-білому форматі.

4. DIY-рішення на базі ESP32-CAM і термопринтера

Серед інженерних спільнот, таких як Hackaday, GitHub чи Tindie, є приклади саморобних проектів на базі ESP32-CAM, які фотографують і друкують зображення через TTL або USB-термопринтери (наприклад, серії 58 мм або 80 мм). У таких проектах часто використовуються алгоритми дізерингу для покращення якості друку на монохромному термопапері. Це рішення є бюджетним, відкритим і легко модифікованим, що робить його привабливим для ентузіастів та освітніх проектів.

2.2. Огляд існуючих мережевих рішень для автономних IoT-пристроїв

Сучасні автономні пристрої, що працюють у сфері Інтернету речей (IoT), потребують ефективних і гнучких мережевих рішень, здатних забезпечити швидку та безпечну передачу даних, зручне управління та мінімальні вимоги до зовнішньої інфраструктури. У зв’язку з цим розробники IoT-пристроїв мають змогу обирати серед низки перевірених підходів, які адаптовані до різних апаратних ресурсів, умов експлуатації та типів завдань. У цьому підрозділі розглянуто найбільш поширені мережеві рішення, які використовуються в автономних системах, подібних до пристроїв на базі ESP32-CAM.

1. Wi-Fi Access Point Mode (AP Mode) - один з найпопулярніших підходів, особливо для компактних пристроїв, які повинні працювати автономно. У цьому режимі мікроконтролер сам створює точку доступу Wi-Fi, до якої можуть підключатись клієнти — смартфони, планшети або комп’ютери. Це дозволяє реалізувати повноцінний веб-інтерфейс без необхідності зовнішнього маршрутизатора чи Інтернет-з’єднання. Такий підхід використовується, наприклад, у портативних сканерах, 3D-принтерах, фотопринтерах та інших автономних гаджетах.

2. Wi-Fi Station Mode (STA Mode) - у цьому випадку пристрій підключається до наявної мережі Wi-Fi і отримує доступ до локальної інфраструктури або Інтернету. Це дозволяє інтегрувати пристрій у вже існуючу мережу, що не підходить для такого портативного пристрою, як полароїд.

3. REST API на HTTP-сервері - простий у реалізації підхід, що дозволяє клієнту надсилати запити до пристрою за допомогою HTTP. Це рішення чудово підходить для пристроїв із простими веб-інтерфейсами, коли користувач виконує окремі дії — наприклад, зйомку фото, запуск друку або передачу тексту. Саме цей підхід був реалізований у даному проекті для взаємодії з ESP32-CAM через браузер.

З огляду на специфіку проекту, найдоцільнішим виявився підхід на основі Wi-Fi Access Point у поєднанні з HTTP REST API, що дозволяє створити простий, зрозумілий і незалежний спосіб взаємодії з пристроєм без потреби у зовнішніх серверах чи спеціальному клієнтському ПЗ.

2.3 Аналіз алгоритмів перетворення кольорового зображення у чорнобіле

При розробці пристрою, що друкує зображення на термопринтері, важливим етапом є обробка фото, отриманого з камери, у формат, придатний до друку. Оскільки більшість термопринтерів підтримують лише монохромний режим (чорне або біле), виникає потреба у спеціальних алгоритмах, які дозволяють перетворити повнокольорове зображення в оптимізовану чорно-білу версію, зберігаючи при цьому якнайбільше візуальної інформації.

Звичайне порогове перетворення (thresholding), при якому всі пікселі яскравіші за певне значення стають білими, а темніші — чорними, є надто грубим методом.

Приклад такого перетворення (thresholding) наведено нижче:

Рис. 2.1. Демонстрація роботи tresholding

Він часто призводить до втрати деталей та неприроднього вигляду зображення. Для вирішення цієї проблеми застосовуються алгоритми дизерингу (англ. dithering), які дозволяють імітувати різні відтінки сірого за допомогою комбінацій чорних і білих пікселів.

1. Алгоритм Floyd-Steinberg Dithering

Найпоширенішим і найефективнішим алгоритмом дизерингу є алгоритм Флойда-Стейнберга. Його суть полягає в тому, щоб після перетворення поточного пікселя на чорний або білий, помилку квантування (різницю між оригінальним значенням пікселя та новим) розподілити на сусідні пікселі, які ще не були оброблені. Це створює ілюзію градацій сірого при виведенні на чорно-білий пристрій.

Формально розподіл помилки відбувається за наступною схемою:

*	7/16	
3/16	5/16	1/16

Така схема дозволяє "розмазати" помилку квантування у напрямку обробки, завдяки чому загальне зображення виглядає плавнішим і реалістичнішим.

Інші алгоритми дизерингу

Окрім Floyd-Steinberg, існує велика кількість альтернативних алгоритмів дизерингу, які використовуються в залежності від вимог до швидкодії, пам’яті, візуальної якості та цільового пристрою. Далі розглянемо найбільш поширені з них:

2. Ordered Dithering (Упорядкований дизеринг)

Цей метод не розподіляє помилку, як Floyd-Steinberg, а застосовує передвизначену матрицю (найчастіше — матрицю Байєра), яка додає періодичний шум у зображення. Порівнюючи яскравість пікселя з відповідним значенням з матриці, можна визначити, чи буде піксель чорним або білим.

Приклад 4×4 матриці Байєра:

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Переваги: висока швидкість, простота реалізації, хороша стабільність на пристроях із обмеженими ресурсами (наприклад, ESP32).

Недоліки: характерні артефакти у вигляді сітки або зернистості; не така реалістична передача градацій, як у помилково-розподілених алгоритмах.

3. Atkinson Dithering

Алгоритм, розроблений Біллом Аткінсоном для комп’ютерів Macintosh. Як і Floyd-Steinberg, Atkinson розподіляє помилку, але на меншу кількість сусідніх пікселів, і в меншій пропорції.

	*	1	1
1	1	1	
	1		

Усі розподілені частки мають однакову вагу (1/8 від помилки).

Переваги: створює легке зерно, м’які переходи; візуально схожий на Floyd-Steinberg.

Недоліки: менш точний, трохи втрачає контраст; дещо "мильний" вигляд на великих деталях.

4. Jarvis, Judice, and Ninke (JJN) Dithering

Один із найточніших алгоритмів дизерингу, що розподіляє помилку на більше число сусідніх пікселів (до 12). Завдяки цьому виходить дуже плавне зображення.

	*	7	5	
3	5	7	5	3
1	3	5	3	1

Переваги: відмінна якість зображення, плавні градації.

Недоліки: значне навантаження на процесор та пам'ять — не підходить для слабких мікроконтролерів.

5. Stucki Dithering

Варіація алгоритму JJN з меншими вагами — призначена для прискорення обробки при збереженні якості.

Схема подібна до JJN, але ваги менші:

	*	8	4	
2	4	8	4	2
1	2	4	2	1

Переваги: хороша візуальна якість, менше спотворень, ніж у JJN, але економічніше.

Недоліки: усе ще досить ресурсомісткий для мікроконтролерів.

6. Burkes Dithering

Більш легка версія JJN, яка зберігає прийнятну якість і знижує навантаження на систему.

	*	8	4	
2	4	8	4	2

Переваги: добра якість при відносно низькій складності.

Недоліки: трохи нижча деталізація в порівнянні з JJN і Floyd-Steinberg.

7. Sierra Dithering

Існує кілька варіантів Sierra дизерингу (Sierra-2, Sierra Lite), які оптимізовані для балансування якості та швидкодії.

Наприклад, Sierra Lite:

	*	2
1	1	

Переваги: компактна реалізація, придатна для мікроконтролерів.

Недоліки: менш плавне зображення, ніж у повноцінних алгоритмах. Порівняльна таблиця:

Алгоритм	Якість	Швидкодія	Споживання пам’яті	Складність реалізації
Floyd-Steinberg	Висока	Середня	Середнє	Середня
Ordered (Bayer)	Середня	Висока	Низьке	Низька
Atkinson	Висока	Висока	Низьке	Низька
Jarvis-Judice-Ninke	Дуже висока	Низька	Високе	Висока
Stucki	Висока	Середня	Високе	Висока
Burkes	Середня	Середня	Середнє	Середня
Sierra Lite	Середня	Висока	Низьке	Низька

У підсумку, при виборі алгоритму для пристрою з ESP32-CAM та термопринтером, найкращими варіантами є Jarvis-Judice-Ninke або Atkinson залежно від пріоритетів: якість чи швидкість.

Враховуючи низьку роздільну здатність камери, а саме 160x120 пікселів – можна подумати, що найкращим вибором буде алгоритм, який якомога краще зберігає і без того, не велику якість. Проте в ході тестувань різниця в алгоритмах Atkinson та Jarvis-Judice-Ninke була незначна, але складність реалізації сильно відрізнялася, за рахунок чого вирішено робити натиск на легкодію та простоту

написання, щоб зекономити оперативну пам’ять мікроконтроллера, Вибір було зупинено на найпопулярнішому алгоритмі Floyd-Steinberg Dithering.

2.4 Реалізація повного електронно кола з автономним живленням

Посилаючись на вміст підрозділу 1.4 «Створення теоретичної схеми пристрою» було вияснено, що потрібна напруга та струм становлять 6-9 вольт та 3 ампера відповідно.

Нижче представлено три реалістичні варіанти автономного живлення, їхню реалізацію, переваги й недоліки:

Варіант 1: Один високоамперний аккумулятор 18650 + підвищуючий модуль

Компоненти:

1) 1 аккумулятор 18650 (≥3000 мА·год, ≥10 А розряд)
2) Підвищуючий перетворювач струму
3) Зовнішній захист від перерозряду
4) Зовнішній пристрій для зарядки акумуляторів.

При простому під’єднанні з переваг є лише простота в створенні та обслуговуванні, але очевидні наступні недоліки**:** Значне нагрівання компонентів при струмах >2 А, Швидкий розряд при високому навантаженні

Зарядка: потрібна спеціальна зарядка з балансуванням або модуль зарядки акумуляторів від звичайного побутового блоку живлення.

Варіант 2: Два аккумулятор и 18650 + BMS з послідовним з'єднанням (2S)

Компоненти:

1) 2 × 18650 (≥2500 мА·год, ≥5 А)
2) балансувальна-плата на 2S

Таке підключення є більш оптимальним адже мінімальна напруга становить 6.0 вольт а номінальна - 7.4 вольт. Балансувальна плата забезпечує безпечний заряд та розряд акумуляторів, а розподілення навантаження на 2 акумулятори збільшить стабільність та термін їхньої дії. З недоліків можна назвати необхідність у звичайному підвищуючому модулі для того, щоб подати напругу на балансувальну (BMS) плату, адже заряджати акумулятори у 2S

компонуванні потрібно мінімум їх номінальною напругою. У даному випадку зарядка має проводитися напругою 8.4 вольт.

Варіант 3: Один літій-полімерний (LiPo) аккумулятор на 7.4 В з високим струмом

Компоненти:

1) LiPo 7.4 В 2200–3000 мА·год з розрядом ≥25C
2) Стандартний балансний коннектор (JST-XH)

Даний варіант є найбільш продуманим та надійним, адже вихідна напруга одразу є підходящою та не панаю при навантаженні. Що є важливим для рівномірного друку на принтері, адже насиченість чорного кольору прямопропорційно залежить від напруги яка подається на вхід живлення принтера. Також такі акумулятори швидко заряджаються. Недоліками можна назвати його ціну. Також часто такий акумулятор потрібно від’єднувати під час заряджання. Також заряджання проходить по спеціальному протоколу, для якого все рівно доведеться мати спеціальні перетворювачі для побутових блоків живлення або спеціального блоку живлення стандартизованого під такі акумулятори.

Проаналізувавши ці варіанти вирішено було скористатися двома аккумулятор ами 18650 та BMS платою з послідовним з'єднанням (2S), а також заряджанням від побутової зарядки 5 вольт через перетворювач на 8.4 вольт. Даний варіант здається найбільш оптимальним по ціні та якості для такого проекту, як полароїд.

Також для демонстрації наявного заряду варто додати відповідний модуль, який показує в поділках заряд від 6 до 8.4 вольта, де 0 або 1 поділка – це 6 і менше вольт, а 4 поділки – 8.4 вольти та більше. Теоретична схема зображена нижче: (рис.2.2).

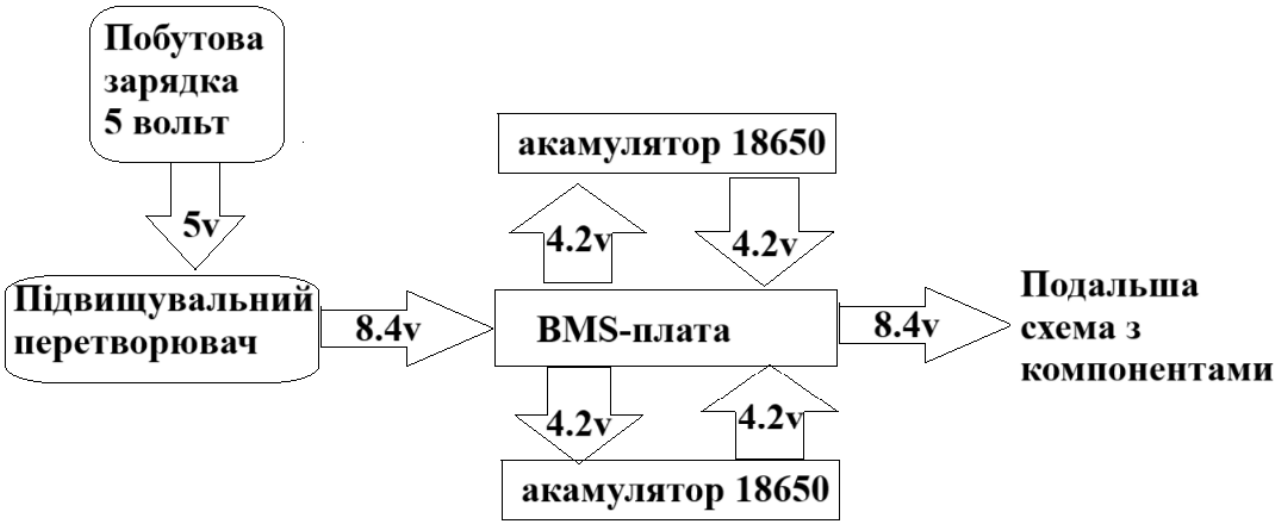

Рис. 2.2. Теоретична схема автономного живлення пристрою

РОЗДІЛ 3
ПРОГРАМНА ТА АПАРАТНА РЕАЛІЗАЦІЯ

3.1 Розробка та реалізація схеми електричного кола проекту

Основуючись на схемах з розділу 1.4. «Створення теоретичної схеми пристрою» і 2.4 «Реалізація повного електронно кола пристрою з автономним живленням», а також документаціях до кожного компонента було створено наступну схему електричного кола:

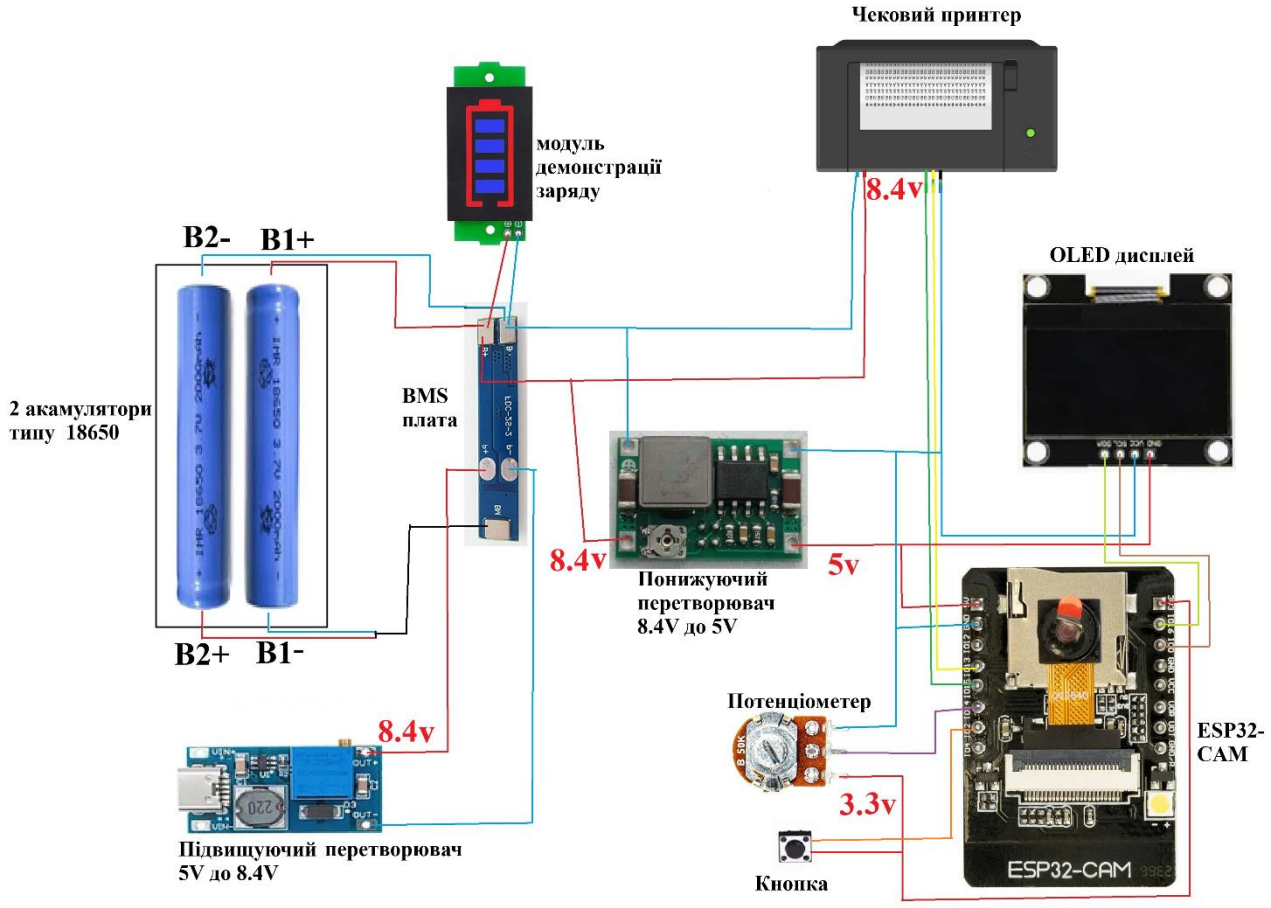

Рис. 3.1. Практична схема готового пристрою

Простеживши за схемою можна виділити і описати 2 наступних режиму роботи:

Заряджання: живлення від побутового блоку живлення від телефону, наприклад, поступає на підвищуючий модуль, після чого живлення з напругою 8.4 вольт подається на BMS-плату, яка стежить за розрядом кожного

акумулятора окремо та підзаряджає більш розряджений або паралельно заряджає 2 акумулятори одночасно, в імпульсному режимі.

Робота пристрою: живлення від контактів двох, послідовно з'єднаних акумуляторів, напругою в 6-8.4 вольти, залежно від рівня заряду, поступає напряму на модуль демонстрації заряду та принтер і на понижуючий перетворювач, який робить з не стабільного живлення від акумуляторів – стабільне живлення напругою в 5 вольт. Ці 5 вольт розходяться на живлення OLED дисплею та ESP-32. І вже завдяки стабілізатору на платі ESP32 на піні «+3.3v» можна отримати підходяще живлекння напругою 3.3 вольти, яке вже передається на кнопку та потенціометр. Даний сигнал можна зчитати логічними пінами без шкоди для електронних компонентів.

Прошу зауважити, що всі лінії gnd об'єднані в одну, що забезпечує стабільну передачу логічного сигналу між електронними компонентами. Також варто зазначити, що підключення логічних пінів на схемі може відрізнятися від реального підключення.

3.2 Реалізація структури проекту

Увесь проект поділено на модулі, кожен з яких відповідає за певний функціонал. Нижче наведено повний перелік та опис таких частин:

1. polaroid.ino – є головним файлом всього проекту. Відповідає за запуск ініціалізаційних методів для інших файлів у функції setup, а також за виклик методів опитування веб-сервера, камери, енергонезалежної пам'яті та органів керування.

2. webcamera.ino – відповідає за методи пов'язані з отриманням фотографій оброблених в imageparse.ino.

3. imageparse.ino – є допоміжним файлом, який місти в собі функції згладжування та дізерингу, які перетворюють зображення у підходяще для виведення.

4. display.ino – містить функції для перетворення обробленого фото з розширенням 160x120 у підходяще 128x64 та відмальовуванням такого зображення на дисплеї.

5. logger.cpp - клас для створення об’єкту logger який, через метод log(“message”); додає повідомлення до буферу, після чого список повідомлень буде виводитися та оновлятися на сторінці веб серверу

6. printer.ino – містить в собі методи ініціалізації принтера, а також методи для перетворення та відправлення даних нра друк

7. webserver.ino - відповідає за розгортання веб сервера та налаштування ендпоїнтів з прив’язаними функціями серед інших файлів.

У наступних підрозділах буде описано роботу кожного модуля та основних його методів більш детально.

3.2.1 Робота polaroid.ino

Після запуску плати, виконується метод setup, який містить в собі ініціалізцію:

1) енергонезалежної пам’яті - EEPROM.begin(mem.blockSize());mem.begin(0, 'a');
2) внутнішнього серверу камери - setupCameraServer();
3) веб-серверу - startWebServer();
4) парсера фотографій - setupTJpg();
5) дисплея - setupDisplay();

Після ініціалізації починає безкінечно виконуватися метод loop(), який виглядає наступним чином:

```
void loop() {
  const mls = millis();
  if (mls - btnPotTmr >= buttonPotentiometerTmrInMlscnds) {
     btnPotTmr = mls;
     handleButtonPot();
  }
```

```
  if (mls - scrnUpdtTmr >= screenUpdateTmrInMlscnds) {
     scrnUpdtTmr = mls;
     takeAndReturnParsedPhotoHex();
  }
  mem.tick();
  handleServer();
}
```

Як видно вище, в методі записується значення millis(), яке є кількістю мілісекунд після старту плати. Після того починаються перевірки на те, чи стало теперішнє значення пройдених секунд більше за значення останнього разу виконання даної частини коду більше, на вказаний поріг спрацьовування.

```
mls - btnPotTmr >= buttonPotentiometerTmrInMlscnds
```

В даному випадку значення buttonPotentiometerTmrInMlscnds рівне 50, що означає, що кожні 50 мілісекунд буде виконуватися умова входу, і буде викликано метод handleButtonPot();. Нижче наведено приклад як працює аний алгоритм на схемі:

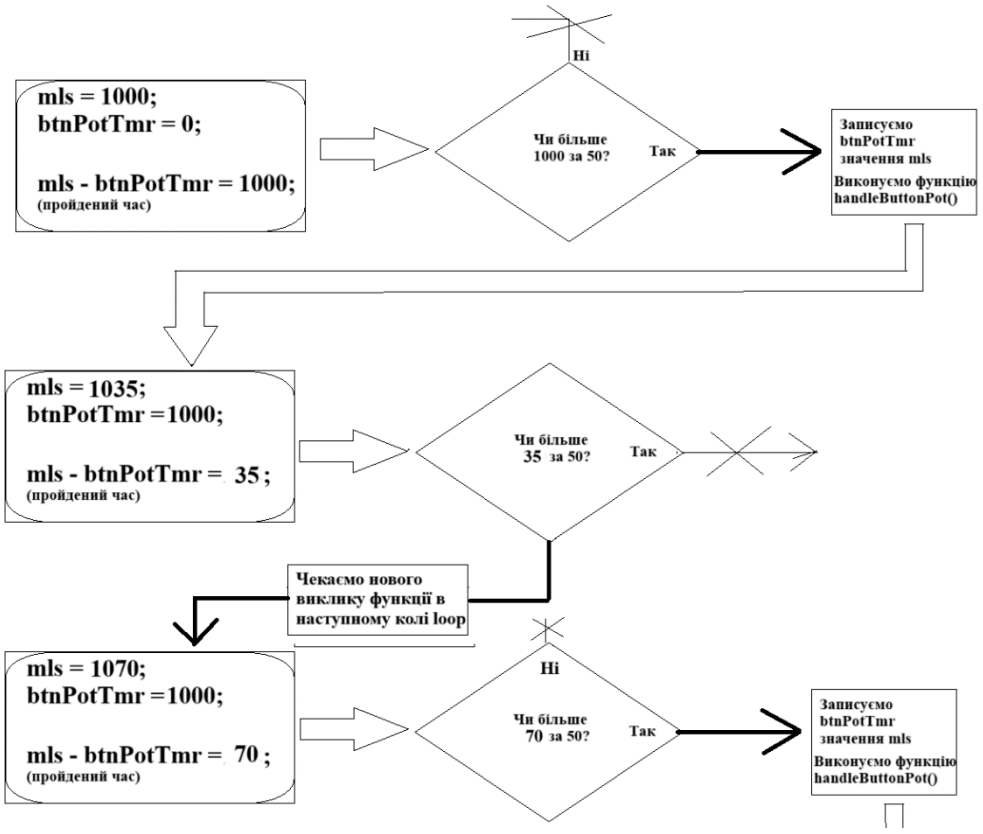

Рис. 3.2. Демонстрація алгоритму інтервалізації потрібних участків коду

Функція handleButtonPot опитує логічний пін потенціометра:

```
int potRaw = analogRead(POT_PIN);
```

Перевіряє чи змінилося значення відносно попереднього виміряного стану хоча б на 10/4095 одиниць:

```
if(abs(prevPotRawValue - potRaw)>10){
  prevPotRawValue=potRaw;
  exposition = map(potRaw, 0, 4095, 0, 124);
}
```

і якщо таки змінилося – оновляє значення prevPotRawValue та значення exposition, яке використовується у файлі imageparse.ino.

Таку перевірку потрібно робити, бо рівень exposition можна міняти програмно з сервера, і щоб не вводити оремо сесійну експозицію і локальну, вирішено оновляти цей параметр лише тоді, коли на потенціометрі відбуваються значущі зміни. Якщо є потенціометр не чіпати, можна буде керувати значенням exposition віддалено через веб-сервер.

Після того, як розібрались з потенціометром – переходимо до визначення стану кнопки: bool currentBtnState = digitalRead(BUTTON_PIN); і якщо розпізнано, що було натиснуто кнопку, і ми бачимо, що перед чим кнопка не була натиснута, камера та принтер не зайняті `if(!buttonPrevPressState && isCameraFree && isPrinterReady)` – тоді робимо знімок і відправляємо його на друк:

```
String hexData = takeAndReturnParsedPhotoHex();
DecodeAndPrintPhoto(hexData);
```

Такий алгоритм дозволяє уникнути постійних викликів функції друку якщо не встигнути відпустити кнопку до наступних викликів методу handleButtonPot().

3.2.2 Робота logger.cpp

Після створення об’єкту класу logger, стає доступна можливість виконати метод `.log();` і передати повідомлення яке буде покладене до буферу LogMessage

всередині об’єкта класу. Розмір даного буферу визначається зарання і залишається не змінним. Повідомлення зберігаються у порядку черги, що означає, якщо буфер перерповнений і метод log() знову викликається – найдавніше повідомлення видаляється, а нове – записується. Для збереження порядку відправлених логів використовується звичайна індексація. Також є відкритий метод getLogsAfter(uint32_t lastId), який повертає всі логи, індекс яких буде вищий за переданий номер. Виглядає цей метод наступним чином:

```
String Logger::getLogsAfter(uint32_t lastId) const {
        String json = "[";
        bool first = true;
        for (size_t i = 0; i < count; ++i) {
          if (buffer[i].id > lastId) {
            if (!first) json += ",";
              json += "{\"id\":" + String(buffer[i].id) + ",\"message\":\"" +
buffer[i].message + "\"}";
            first = false;
          }
        }
        json += "]";
        return json;
      }
```

Як це працює: Наприклад користувач заходить на будь який ендпоїнт на веб-сервері, де увімкнено логування (у даному релізі проекту такими є всі доступні сторінки). Автоматично, після входу на сторінку пристрій починає слати запити на плату, з певним інтервалом, на ендпоїнт, який прив’язаний до методу getLogsAfter(uint32_t lastId). Оскільки при ініціалізації сторінки сесійний параметр lastId =0; - всі доступні повідомлення будуть додані до рядку json та повернуті сторону клієнта, де і будуть оброблені, поділені та виведені у консоль, де користувач зможе їх переглядати. Найбільший індекс повідомлення буде записано до сесійного lastId. Тепер при новому запиті до getLogsAfter(), буде виконано перевірку чи є якісь нові повідомлення. Якщо нових повідомлелень немає - у відповідь буде повернуто пустий масив. Але якщо нові повідомлення, все ж з’явилися – лише їх буде додано до масиву та відправлено на веб-сторінку, де вони будуть так само виведені у консоль.

Даний підхід дозволяє відстежувати N-ну кількість останніх повідомлень у зручному форматі, без необхідності підключатися до serial порту.

Також варто зазначити, що використання класу дозволяє використати переваги інкапсуляції та створити кілька незалежних об’єктів logger, наприклад: loggerCamera, loggerPrinter та loggerDisplay, щоб відслідковувати логи упорядковано і лише на потрібних сторінках.

3.2.3 Робота webcamera.ino

Оскільки ESP модуль є не звичайним а стандартизованим під роботу з камерою – присутні внутрішні методи для ініціалізації та використання камери. Для прикладу так виглядає частина конфігурація камери при ініціалізації

```
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
...........
#define PCLK_GPIO_NUM     22
```

А так виглядає подальше використання цих конфігурацій в методі `setupCameraServer()` , який створює об’єкт `config`

```
Serial.println(">> setup() started");
camera_config_t config;
```

та починає записувати конфігурацію у відповідні поля:

```
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer   = LEDC_TIMER_0;
config.pin_d0       = Y2_GPIO_NUM;
config.pin_d1       = Y3_GPIO_NUM;
config.pin_d2       = Y4_GPIO_NUM;
config.pin_d3       = Y5_GPIO_NUM;
config.pin_d4       = Y6_GPIO_NUM;
config.pin_d5       = Y7_GPIO_NUM;
config.pin_d6       = Y8_GPIO_NUM;
config.pin_d7       = Y9_GPIO_NUM;
config.pin_xclk     = XCLK_GPIO_NUM;
config.pin_pclk     = PCLK_GPIO_NUM;
config.pin_vsync    = VSYNC_GPIO_NUM;
config.pin_href     = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn     = PWDN_GPIO_NUM;
```

```
  config.pin_reset    = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  config.frame_size   = FRAMESIZE_QQVGA;
  config.jpeg_quality = 10;
  config.fb_count     = 1;
```

Після чого створену конфігурацію передає далі у метод esp_camera_init який і налаштовує зв’язок з сервером камери і конфігурує все під роботу.

Після того, виводиться результат розпізнавання:

```
 Serial.println(">> Initializing camera...");
 esp_err_t err = esp_camera_init(&config);
   if (err != ESP_OK) {
   Serial.printf("Camera init failed with error 0x%x", err);
   return;
 }
 Serial.println(">> Camera init success");
}
```

Після ініціалізації можна починати роботу з камерою. Усі методи, які відповідають за отримання фотографії використовують запит на метод takeAndReturnParsedPhotoHex(). Працює він наступним чином:

Обнуляємо закешований фрейм, щоб мати останній успішний кадр

```
  camera_fb_t *fb = esp_camera_fb_get();
  if (fb) esp_camera_fb_return(fb);
```

Робимо запит на отримання нового знімку:

```
  fb = esp_camera_fb_get();
  delay(10);
  if (!fb) {
    Serial.println("Camera capture failed");
    return "";
  }
Serial.println("Camera capture ready");
```

Якщо знімок отримано – починаємо його обробку через метод файлу imageparse.ino – decodeAndProcessImage() , а буфер з вже обробленим зображенням пакуємо в hex рядок:

```
  decodeAndProcessImage(fb->buf, fb->len);
```

```
  esp_camera_fb_return(fb);
Serial.println("Starting parsing to hex");

  String hex;
  for (int i = 0; i < sizeof(monoBitmap); i++) {
    if (monoBitmap[i] < 0x10) hex += "0";
    hex += String(monoBitmap[i], HEX);
  }
  hex.toUpperCase();
Serial.print("hex is ready: ");
```

Відправляємо на OLED дисплей та повертаємо туди, звідки відбувся виклик методу:

```
decodeAndDisplayImage(monoBitmap, sizeof(monoBitmap));
 return hex;
```

3.2.4 Робота imageparse.ino

Файл imageparse.ino виконує одну з ключових функцій проекту — обробку зображення, зокрема JPEG-файлу, з подальшим перетворенням його в чорно-білу карту пікселів, придатну для друку на чековому термопринтері. Оскільки принтер не підтримує кольорову або відтінкову графіку, важливо застосувати дизеринг (dithering), аби створити ілюзію сірого кольору з використанням лише чорних і білих пікселів.

Ця бібліотека дозволяє декодувати зображення у форматі JPEG безпосередньо в пам’яті мікроконтролера (ESP32/ESP8266). Вона обробляє байтовий масив JPEG і викликає користувацьку функцію drawBlock() для кожного сегменту зображення. Це дозволяє реалізувати власну обробку пікселів — у нашому випадку, алгоритм Dithering.

Вбудовані мікроконтролери, такі як ESP32 або ESP8266, мають обмежені обчислювальні ресурси та пам’ять. Розпакування JPEG вручну або попіксельна обробка зображень великого розміру — складна задача. Бібліотека TJpg_Decoder ефективно декодує JPEG «на льоту» без великої RAM, дозволяє реалізувати власну обробку через drawBlock(); що дозволило інтергувати Dithering в бібліотеку, а також вона є добре оптимізована під архітектуру ESP.

Буфери помилки errorBuffer[] та nextErrorBuffer[] Реалізують алгоритм дизерингу Флойда-Стейнберґа, який поширює помилку квантування на сусідні пікселі для створення «візуально сірого» ефекту, зберігаючи тільки чорно-білі точки.

Функція setupTJpg() налаштовує декодер та передає функцію drawBlock() як колек для обробки кожного блоку декодованих пікселів:

```
void setupTJpg(){
  TJpgDec.setJpgScale(1);  // No scaling
  TJpgDec.setCallback(drawBlock);
}
```

Основною функцією є decodeAndProcessImage, яка приймає JPEG-зображення у вигляді байтового масиву (jpgData) та довжину. Вона ініціалізує та викликає викликає TJpgDec.drawJpg() для обробки всього зображення і вже саме ця функція, через вказаний колбек виконує обробку зображення. Обробка проходить приблизно за таким алгоритмом:

1) Отримує блоки пікселів у форматі RGB565.
2) Кожен піксель перетворюється у градацію сірого за стандартною формулою:
   gray = 0.3 * R + 0.59 * G + 0.11 * B
3) З урахуванням накопиченої помилки дизерингу, визначається остаточне значення (0 або 255).
4) Якщо нове значення — чорне (0), піксель у monoBitmap очищається відповідним бітом.
5) Розрахована помилка поширюється на сусідні пікселі згідно зі схемою Флойда-Стейнберґа:
6) Після кожного рядка масив помилок оновлюється: nextErrorBuffer в errorBuffer.
7) Варто зазначити, що є параметр exposition, який може бути налаштований через сервер та через обертання ручки потенціометра. Цей параметр дозволяє налаштувати яскравість зображення перед дизерингом. Значення експозиції додається до градації сірого, компенсуючи недоекспоновані або переекспоновані знімки. Це особливо корисно, якщо камера або умови освітлення змінюються.

3.2.5 Робота display.ino

Файл display.ino відповідає за виведення чорно-білих зображень на OLED-дисплей на основі однобітної матриці пікселів, яка попередньо формується з JPEG-зображення. Для реалізації цього відображення використовується бібліотека Adafruit_SH1106G, сумісна з SH1106 або SSD1306 дисплеями розміром 128×64 пікселі.

При старті плати в методі setup викликається метод setupDisplay(), який відкриває шину спілкування Wire з дисплеєм:

```
Wire.begin(SCKpin, SDApin);
```

і якщо ініціалізація проходить успішно – відбувається очистка дисплею:

```
display.clearDisplay();
display.display();
```

Для виведення на дисплей фото, яке було розпізнане камерою використовується метод decodeAndDisplayImage(), який з отриманого буфера 160x120 з камери перетворює в підходящий 128x64 для дисплея.

Досягається це стандартною формулою масштабування, а саме спершу визначенням коефіцієнту стиснення:

```
sourceX = targetX * (SOURCE_WIDTH / TARGET_WIDTH)
sourceY = targetY * (SOURCE_HEIGHT / TARGET_HEIGHT)
```

І після побітове зчитування й установку пікселів у новий масштабований масив:

```
for (int targetY = 0; targetY < TARGET_HEIGHT; targetY++) {
   for (int targetX = 0; targetX < TARGET_WIDTH; targetX++) {
      int sourceX = (int)(targetX * xScale);
      int sourceY = (int)(targetY * yScale);
     int byteIndex = (sourceY * (SOURCE_WIDTH / 8)) + (sourceX / 8);
      int bitIndex = sourceX % 8;
      if (monoBitmap[byteIndex] & (1 << (7 - bitIndex))) {
          int newByteIndex = (targetY * (TARGET_WIDTH / 8)) + (targetX / 8);
         int newBitIndex = targetX % 8;
```

```
            newMonoBitmap[newByteIndex] |= (1 << (7 - newBitIndex));
          }
        }
      }
```

Ну і оброблений масив відправляється на дисплей, який попередньо очищається:

```
    display.clearDisplay();
        display.drawBitmap(0, 0, newMonoBitmap, 128, 64, SH110X_WHITE);
    display.display();
```

3.2.6 Робота printer.ino

Даний модуль вимагає лише виклику методу printer.begin(); для коректної ініціалізації принтера і після можна починати спокійно друкувати. Для взаємодії використовується бібліотека Adafruit_Thermal.h .

Для початку друкування є кілька функцій, серед яких є метод printTextWithFont()

Даний метод приймає 2 параметра: повідомлення та розмір. На вибір є три формати:

1) малий (приблизно відповідає 12px)
2) середній (приблизно відповідає 24px)
3) великий (приблизно відповідає 36px)

Дані формати встановлюється цифрами 1, 2, 3 відповідно:

```
  printer.setFont('A');
  switch (fontOption) {
   case 1:
    printer.setSize('S');  // Малий
    break;
   case 2:
    printer.setSize('M');  // Середній
    break;
   case 3:
    printer.setSize('L');  // Великий
    break;
   default:
    printer.setSize('S');
    break;
  }
```

А після встановлення шрифту просто викликаємо друк:

```
printer.justify('L');
printer.println(F(message));
printer.feed(2);
```

Другим методом є printImage() який основуючись на розмірах отриманої картинки збільшує її максимально, просто повторивши кожен піксель по ширині і висоті N-ну кількість раз, яка записується як koof= totalWidth/originalWidth:

```
for(int i = 0; i<koof;i++)
{
 uint16_t outX = x * koof + (totalWidth-originalWidth)/2);
 uint8_t byteIndex1 = outX / 8;
 uint8_t bitPos1 = 7 - (outX % 8);
 lineBuffer[byteIndex1] |= (1 << bitPos1);

 for(int j = 1; j<koof;j++){
  uint16_t outXN = outX + j;
  uint8_t byteIndexN = outXN / 8;
  uint8_t bitPosN = 7 - (outXN % 8);
  lineBuffer[byteIndexN] |= (1 << bitPosN);
 }
}
```

та заповнивши краї білим кольором, щоб відцентрувати зображення – робиться це простою формулою:

```
uint16_t outX = x * koof + (totalWidth-originalWidth)/2);
```

Після весь оброблений буфер виводиться на друк:

```
printer.printBitmap(totalWidth, 1, lineBuffer);
printer.feed(3);
```

3.2.7 Робота webserver.ino

Найбільшим, найважливішим та найцікавішим модулем проекту є webserver.ino. Адже тут відбувається створення того функціоналу, якого не вистачає у звичайних покупних пристроях, а саме веб-інтерфейсу.

При запуску плати одразу ж викликається метод startWebServer() який спершу дістає з енергонезалежної пам’яті параметри ssid та password та якщо вони пусті – заповнює їх стандартними даними:

```
if(data.wifiSSID.length()<=0){
  Serial.print("local wifi ssid is empty. Setting default ");
  data.wifiSSID = "ESP POLAROID";
  ssid = data.wifiSSID;
  Serial.println(ssid);
  mem.updateNow();
}
if(data.wifiPASS.length()<=0){
  Serial.print("local wifi password is empty. Setting default ");
  data.wifiPASS = "12345678";
  password = data.wifiPASS;
  Serial.println(password);
  mem.updateNow();
}
```

Тепер, коли у змінних `ssid` та `password` є дані – вони передаються до стандартного методу бібліотеки WiFi.h `WiFi.softAP(ssid, password);` , який відкриває точку доступу та дозволяє іншим пристроям з модулем wifi доєднуватися до мережі.

Мережа ж отримує вказану назву та пароль, щоб дані з камери не потрапили в чужі руки.

Також тут налаштовуються ендпоїнти сервера, на які ми звертатимемося та отримуватимемо необхідні дані. Кожному енпоїнту відповідає та, чи інша функція:

```
server.on("/get-hexphoto", HTTP_GET, [](){
  String hexData = takeAndReturnParsedPhotoHex();
  if (hexData.length() == 0)
    server.send(500, "text/plain", "Failed to capture image");
  else
    server.send(200, "text/plain", hexData);
});
server.on("/set-exposition", SetExposition);
 server.on("/get-exposition", HTTP_GET, GetExposition);
server.on("/update-wifi-data", UpdateWifiData);
server.on("/get-logs", GetLogs);
```

```
server.on("/", HTTP_GET, []() {server.send(200, "text/html", RenderPage(handle_printer));});
server.on("/photo", HTTP_GET, []() {server.send(200, "text/html", RenderPage(handle_photo));});
server.on("/settings", handle_settings);
```

Серед наведених є прямий поділ на:

1. Виконуючі функції: set-exposition, get-exposition, update-wifi-data, get-logs

та хендлери сторінок, які повертають html код для сторінки: /, photo, settings

Після чого запускається сервер командою server.begin(); який починає слухати запити.

Серед виконавчих функцій немає різноманіття, вони всі виконують схожу роль і працюють за схожим принципом, тому нижче наведено розбір лише методу GetLogs() який викликається ендпоїнтом /get-logs.

Коли клієнт звертається до сервера за логами, у запиті передається аргумет “lastId”, який буде оброблено на стороні сервера:

```
if (server.hasArg("lastId")) {
  uint32_t lastId = (uint8_t) server.arg("lastId").toInt();
```

і якщо такий аргумент знайдено і розпізнано – повернеться результат методу logger.getLogsAfter(). З статусом 200 – ОК:

```
server.send(200, "application/json", logger.getLogsAfter(lastId));
```

Якщо ж не було такого параметра – повернеться помилка зі статусом 400 – BadRequest та деталізованою помилкою:

```
server.send(400, "text/plain", "Missing 'lastId' parameter");
```

2. Хендлери: цікавішою частиною є хендлери сторінок, які повертають код HTML, CSS розмітки та додають скрипти до сторінок. Для прикладу розглянемо ендпоїнт «/photo» який веде на сторінку фотографування.

Під час налаштування ендпоїнтів сервера – у відповідність до хендлерів, таких як фото – ставиться метод RenderPage() який повертає оброблену сторінку

```
server.on("/photo", HTTP_GET, []() {server.send(200, "text/html", RenderPage(handle_photo));});
```

Метод RenderPage() просто додає до тексту кожної сторінки верхнє поле з кнопками – navMenu та скрипт для отримання логування logger_script . Виглядає він наступним чином:

```
String RenderPage(const char* bodyContent) {
  return String(navMenu) + bodyContent + logger_script;
}
```

Для економії оперативної пам’яті – всі блоки з великим текстом без змінних – записується у звичайну пам’ять пристрою, як частина програми, досягається це використанням оператора PROGMEM.

Блок меню, як і блоки сторінок з розміткою сторінки не несуть в собі жодної логіки, на відміну від скриптів.

Для прикладу - logger_script працює наступним чином: кожні 3 секунди відбувається запит на ендпоїнт get-logs де як параметр передається значення lastId. Спершу на стороні клієнта воно рівне 0:

```
<script>
let lastId = 0;
setInterval(() => {
  fetch(`/get-logs?lastId=${lastId}`)
```

Після отримання відповіді – парсимо її і дістаємо отримані повідомлення:

```
    .then(response => response.json())
    .then(logs => {
```

Якщо ж повідомлення були а не прийшов пустий масив – відбувається перебирання масиву і поступове виведення нових повідомлень в консоль.

```
      logs.forEach(log => {
        console.log(`[${log.id}] ${log.message}`);
```

Після цього найбільше значення lastId записується у змінну щоб на наступному циклі не отримувати ті ж повідомлення, що були отримані на даному ітераційному циклі:

```
          lastId = Math.max(lastId, log.id);
```

Якщо ж все ж так сталося, що відбулася помилка – вона буде безпечно оброблена іі виведена в консоль:

```
    .catch(err => console.error("Log fetch error", err));
}, 3000); // every 3 seconds
</script>
```

Подібним чином працюють скрипти для отримання фото з камери, але на відміну від даного скрипта – після отримання фото – hex записується у об’єкт canvas:

```
for (let y = 0; y < height; y++) {
  for (let bx = 0; bx < byteWidth; bx++) {
    const b = bytes[idx++];
    for (let bit = 0; bit < 8; bit++) {
      const x = bx * 8 + bit;
      const pi = (y * width + x) * 4;
      const isBlack = (b & (1 << (7 - bit))) === 0;
      const col = isBlack ? 0 : 255;
      img.data[pi] = col;
      img.data[pi + 1] = col;
      img.data[pi + 2] = col;
      img.data[pi + 3] = 255;
    }}
}
ctx.putImageData(img, 0, 0);
```

При натисканні кнопки фотографувати – відправляється стандартний запит на ендпоїнт «/take-photo», прив’язаний до того ж методу, що й реальна кнопка:

```
server.on("/take-photo", TakeAndPrintPhoto);
```

Якщо було вибрано одну з кнопок для відкладеного фото – запит «/take-photo» виконається з затримкою у вказаний час:

```
function delay(ms) { return new Promise(resolve => setTimeout(resolve, ms));}
async function takePhotoWithDelay() {
  const takePhotoButton = document.getElementById('takePhotoBtn');
  takePhotoButton.disabled = true;
  await delay(delayValue*1000);
  fetch("/take-photo")
  .then(response => response.text())
  .then(data => console.log("Response: " + data));
```

```
  takePhotoButton.disabled = false;
}
```

3.3. Проектування та моделювання корпусу

Під час проектування корпусу було очевидно, що найбільша частина має бути в основному корпусі а для збереження мотивів полароїду, камеру і плату було вирішено розмістити зверху. Також було передбачено розміщення органів керування у зручних місцях, виведення дисплею та індикатора заряду назовні а також виділено місце під акумулятори та інші модулі.

Для зручності експортування моделювання проводилося на платформі ThinkerCad. В результаті було змодельовано такий корпус:

Рис. 3.3. 3D модель головного корпусу пристрою

Для правильного закріплення камери у вертикальному положенні було створено рухомий об’єктив, який кріпиться спереду плати та налаштовується на потрібну висоту, щоб не перегинався шлейф: (рис.3.4),

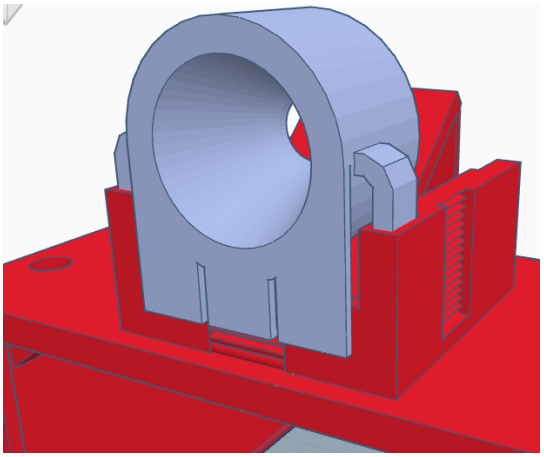

Рис. 3.4. 3D модель рухомого об’єктива пристрою

Для зручного доступу до SD карти – вирішено зробити верхню декоративну кришку модульною: (рис.3.5),

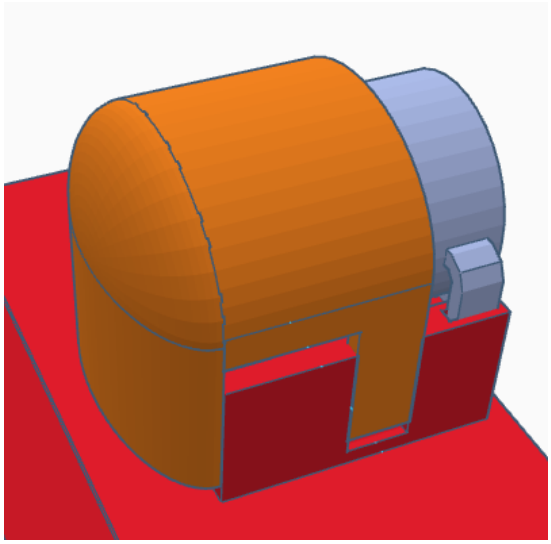

Рис. 3.5. 3D модель рухомої декоративної накладки об’єктиву

Окремо було створено нижню кришку (дно), кнопку та колесико потенціометра. Закінчений вигляд корпусу проекту у повному зборі виглядає так: (рис.3.6),

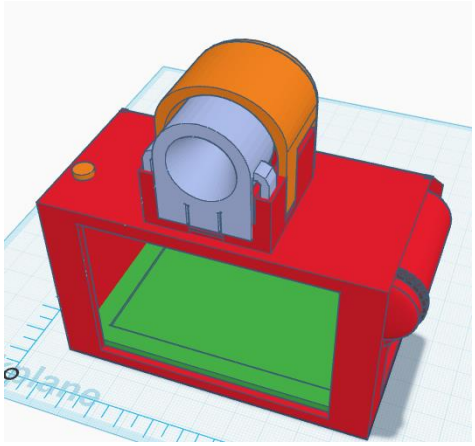

Рис. 3.6. 3D модель пристрою в повному зборі

РОЗДІЛ 4
ТЕСТУВАННЯ І ДЕМОНСТРАЦІЯ РОБОТИ ПРИСТРОЮ

4.1. Тестування роботи пристрою в оффлайн режимі

Перший режим роботи, який передбачався з самого початку – є фотографування і друк моментальних знімків, а також регулювання експозиції ручкою потенціометра. Доступні такі можливості:

1) Створення моментальних фотографій.

Після увімкнення пристрою і ініціалізації компонентів на екрані починає постійно оновлюватися обрізана картинка того, що бачить камера: (рис.4.1),

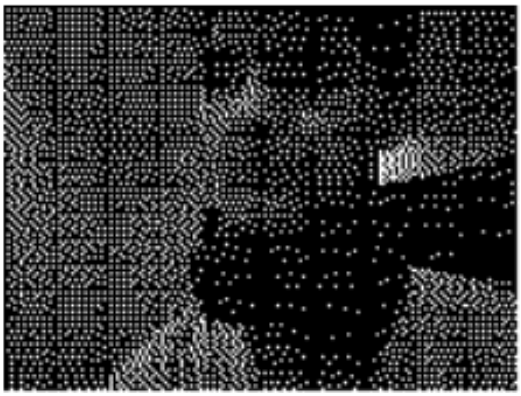

Рис. 4.1. Фото виведення зображення на OLED дисплей

При натисненні на кнопку – відбувається фотографування та отримане фото відправляється на друк, принтер друкує фото, яке поступово виїжджає спреду. Результат такого друку наведено нижче, (рис.4.2),:

Рис. 4.2. Надруковане на чеку моментальне фото з камери

2) Налаштування експозиції.

Після запуску пристрій одразу ж починає вичитувати значення з Потенціометра, тому якщо зйомка проходитиме в однакових умовах освітленості – можна не чіпати ручку потенціометра.

Для демонстрації зміни ефекту візьмемо фото екрану при значенні експозиції приблизно 80 одиниць (~0.7 повного оберту коліщатка потенціометра): (рис. 4.3).

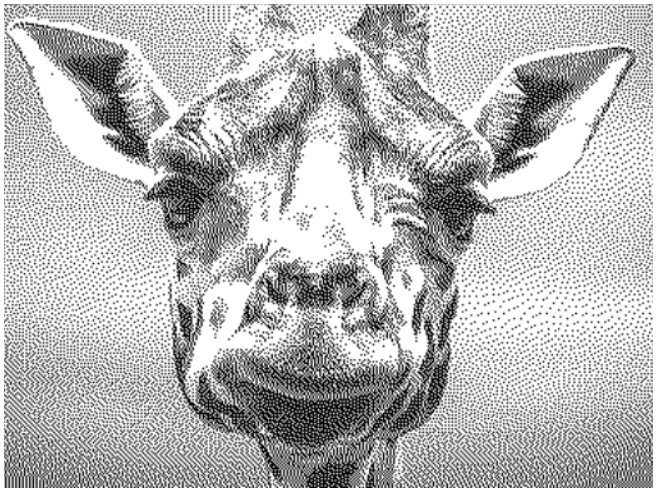

Рис. 4.3. Фото картинки з експозицією вище потрібної

Тепер не змінюючи ракурс і освітлення змінимо експозицію приблизно до 30 одиниць (~0.3 повного оберту коліщатка потенціометра) і звернемо увагу на дисплей:

Рис. 4.4. Фото картинки з експозицією нище потрібної

Як видно, експозиція сильно впливає на якість фотографій, але вона є необхідною про виборі локації і її освітленості. Щоб зберегти обриси та деталі об’єкту зйомки в напівтемряві або ж на переосвітленій області.

4.2. Тестування роботи пристрою в онлайн режимі

Другий режим роботи, який є основним покращенням для фотоапарату з чековим принтером – фотографування та друкування через підключення смартфону до проекту. Після запуску пристрій одразу ж починає створювати точку доступу WiFi. Якщо доєднатися до неї і перейти на стандартнй IP адрес 192.168.1.4 – користувача буде переслано на головну сторінку веб-інтерфейсу камери, де користувач може виконувати різні дії для фотографування, друкування та налаштування точки доступу. Нижче наведено функціонал доступний для онлайн режиму:

1) Перегляд і створення фотографій через смартфон.

Після переходу на сторінку Photo – починається постійне оновлення зображення з камери. Для відправлення фотографії на друк – потрібно натиснути кнопку Take photo. При необхідності – можна змінити експозицію повзунком праворуч, або ж вистачити затримку перед виконанням фото на вказаний період натиснувши на відповідну кнопку ліворуч:

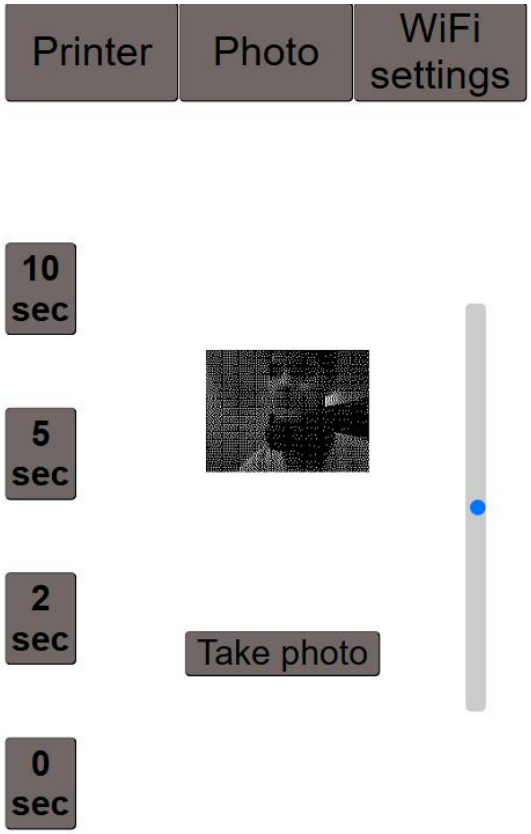

Рис. 4.5. Скріншот сторінки Photo

2) Друкування через смартфон.

При переході на сторінку Printer можна побачити поле введення тексту для повідомлення. За допомогою слайдера – можна змінити розмір шрифта від маленького до великого. Для відправлення на друк даного введеного повідомлення потрібно натиснути кнопку Print:

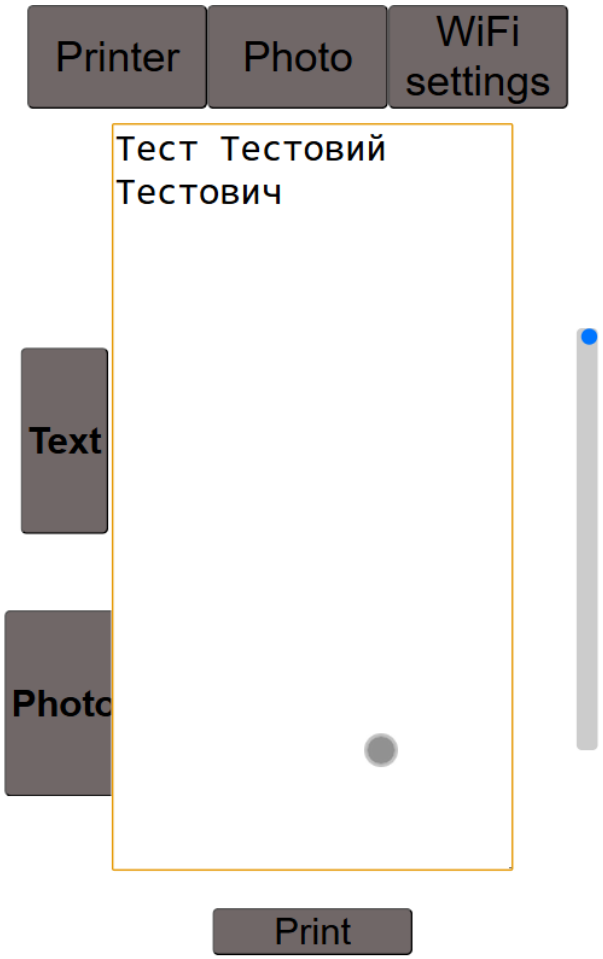

Рис. 4.6. Скріншот сторінки Printer в режимі Text

Якщо перебуваючи на сторінці Printer натиснути кнопку Photo ліворуч від поля внесення – вміст сторінки оновиться наступним чином: для завантаження фото на сторінку потрібно натиснути кнопку «Вибрати файл» та обрати фотографію на пристрої. Обрана фотографія буде оброблена алгоритмом дизеринг автоматично.

Є можливість змінити експозицію посунувши слайдер праворуч, а також - відправити фото на друкування, натиснувши кнопку Print.

Приклад завантаженого фото наведено нижче:

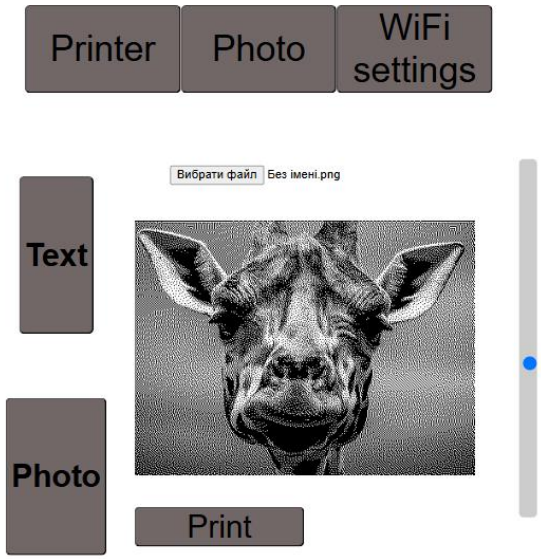

Рис. 4.7. Скріншот сторінки Printer в режимі Photo

3) Налаштування точки доступу через смартфон.

Дана сторінка призначена для зміни імені та паролю точки доступу, яку створює пристрій при ввімкнені. Очевидно. Що для зміни цих параметрів потрібно ввести їх в поля SSID та PASS і після того натиснути кнопку «Apply». Фото даної сторінки наведено нижче:

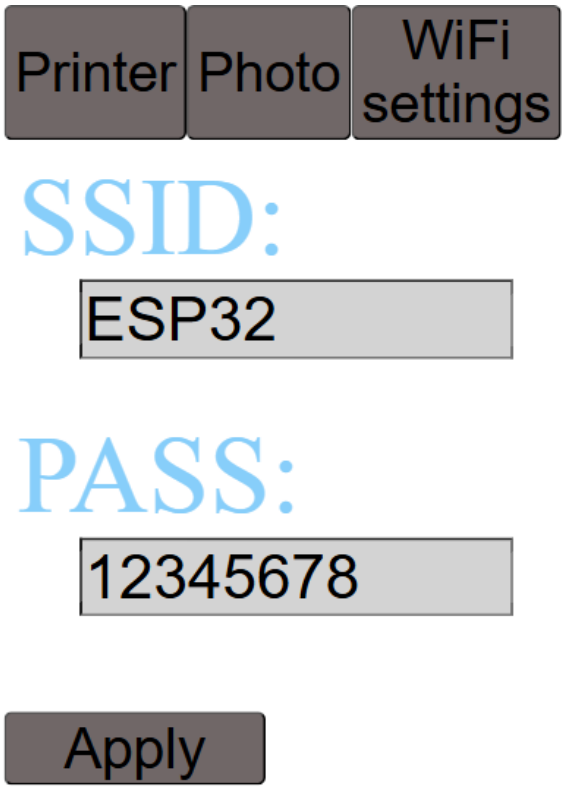

Рис. 4.8. Скріншот сторінки WiFi settings

Якщо після натиснення кнопки «Apply» - надпис кнопки не змінився на success, значить стався якийсь збій і потрібно дивитися логи. Проте якщо все ж надпис змінився на success – параметри були оновлені і при наступному запуску точки доступу – вона запуститься з вказаними ім’ям і паролем.

ВИСНОВКИ

У даній роботі було здійснено повноцінне дослідження та реалізацію проекту пристрою для друку зображень з камери ESP32-CAM на чековому принтері. Основною метою стало створення прототипу, який здатний фіксувати зображення, обробляти його у форматі однобітної графіки (з використанням алгоритмів дизерингу) та здійснювати його фізичний друк за допомогою малогабаритного термопринтера.

Проведено аналіз існуючих рішень, які дозволяють реалізувати вивід зображень на чекові принтери, включаючи комерційні продукти (наприклад, Polaroid Go, PeriPage, Paperang) та відкриті DIY-реалізації. Це дало змогу визначити актуальні підходи, оцінити складність реалізації таких рішень та їх можливості в умовах обмеженого ресурсу мікроконтролерів.

Значну увагу приділено дослідженню алгоритмів перетворення кольорового зображення у чорно-біле. Основним обраним методом став дизеринг, зокрема алгоритм Флойда-Стайнберга, що забезпечує імітацію напівтонових відтінків шляхом правильного розсіювання помилки. Також проаналізовано альтернативні методи дизерингу, що дозволяє більш гнучко адаптувати зображення під характеристики принтера та мікроконтроллера.

Реалізовано зручний веб-інтерфейс який дає змогу вільно користуватися пристроєм у побутових цілях, межі яких оприділяються лише фантазією автора та шириною чекової стрічки, адже реалізована можливість друкування повідомлень та будь яких фотографій дає змогу користуватися принтером не маючи спеціальних навичок за спиною.

Крім того, проведено аналіз різних варіантів живлення для автономної роботи пристрою, включаючи рішення на базі одного чи двох аккумулятор ів 18650 або ж літій-полімерного аккумулятор а. Оцінено зручність заряду, компактність та відповідність вимогам споживаного струму, що дозволило обрати найбільш збалансований варіант живлення для польових умов експлуатації. У підсумку, виконана робота доводить можливість створення малогабаритного, автономного пристрою для захоплення та друку зображень у

реальному часі, що може бути використаний як основа для подальших розробок у сфері портативної друкарської техніки, DIY-фотографії, освітніх проектів для поширення технології фотографування і парсингу фото у маси та створення декоративних інсталяцій.

Робота не лише досягла поставленої мети бути прототипом, але й завдяки корпусу і автономності змогла стати повноцінним пристроєм, від якого не відмовиться як дитина, так і дорослий

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Протоколи шлюзу, Обчислювальна потужність і пам'ять, Пристрої ESP32 IoT. URL: https://www.dusuniot.com/uk/blog/how-esp32-iot-is-changing-the-game-for-connected-devices/ (Дата звернення: 08.05.2025).
2. Скільки струму споживають OLED-дисплеї. URL: https://bitbanksoftware.blogspot.com/2019/06/how-much-current-do-oled-displays-use.html (Дата звернення: 16.05.2025).
3. Скільки струму споживають камери і модулі ESP32Cam через Wi-Fi. URL: https://www.reddit.com/r/esp32/comments/jy8inw/whats_the_minimum_current_my_power_supply_should/ (Дата звернення: 20.05.2025).
4. Згладжування зображення: одинадцять алгоритмів та вихідний код URL: https://tannerhelland.com/2012/12/28/dithering-eleven-algorithms-source-code.html (Дата звернення: 21.05.2025).
5. Повний посібник по роботі ESP32-Cam. URL: https://www.diyengineers.com/2023/04/13/esp32-cam-complete-guide/ (Дата звернення: 08.05.2025).
6. Панельний принтер EM5822. Технічна документація. URL: https://www.mantech.co.za/datasheets/products/em5822-241104a.pdf?srsltid=AfmBOopORJcIuQNfqotVBoXOShFkVlh9P9ECa5MGAkVesVXx4l7md7UX (Дата звернення: 25.05.2025).
7. Деталі роботи та приклади коду Dithering алгоритмів. URL: Ulichney, Robert. Digital Halftoning. MIT Press, 1987. (Дата звернення: 29.05.2025).
8. Деталі роботи OLED дисплеїв через I2C. URL: https://www.instructables.com/OLED-I2C-Display-ArduinoNodeMCU-Tutorial (Дата звернення: 30.05.2025).
9. Деталі роботи та приклади коду Dithering алгоритмів. URL: Floyd, R. W., and Steinberg, L. An adaptive algorithm for spatial greyscale. Proceedings of the Society for Information Display, 1976. (Дата звернення: 08.06.2025).
10. Алгоритм і приклад роботи TJpg_Decoder. URL: https://github.com/Bodmer/TJpg_Decoder (Дата звернення: 10.06.2025).