Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Програмно-технічний засіб вбудованої відеосистеми з ESP32 та OV7670 із передаванням даних через WebSocket
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КІ 22009.22.03.03 ПЗ

Виконав здобувач IV курсу, група КІ2-22-3	_____	Владислав ВИСОЧИНСЬКИЙ
	Підпис	Ініціали, прізвище
Керівник Науковий ступінь, учене звання	_____ Підпис	Сергій ЛИСЕНКО Ініціали, прізвище
Нормоконтролер Науковий ступінь, учене звання	_____ Підпис	Сергій ЛИСЕНКО Ініціали, прізвище
До захисту допускаю: завідувач кафедри КІІС «___» _червня_ 2026 р. дата	_____ підпис	Ольга ПАВЛОВА Ініціали, прізвище

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС

_____Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Височинський Владислав В'ячиславович
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Програмно-технічний засіб вбудованої відеосистеми ESP32 та OV7670 із передаванням даних через WebSocket

Керівник проєкту (роботи) Лисенко Сергій Миколайович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз сучасних вбудованих відеосистем на базі мікроконтролерів та постановка задачі розробки програмно-технічного засобу

Проєктування архітектури та обґрунтування технічних рішень програмно-технічного засобу вбудованої відеосистеми на базі ESP32

Програмно-апаратна реалізація та експериментальне дослідження вбудованої відеосистеми

з використанням сенсора OV7670 та протоколу WebSocket

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

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 10 » 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проєкту (роботи)	Термін виконання етапів проєкту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об’єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – вибір компонентів для виконання поставленої задачі проєкту	01.04.2026	виконано
5	Робота над розділом 3 – проєктуваня та програмно-апаратна реалізація вбудованої відеосистеми на базі ESP32 та OV7670	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ Владислав ВИСОЧИНСЬКИЙ
Підпис Ім’я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ Сергій ЛИСЕНКО
Підпис Ім’я, ПРІЗВИЩ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КІ 22009.22.03.03 ПЗ	Пояснювальна записка	57		
			Графічні матеріали			
2		КІ 22009.22.03.03 ПЗ	Архітектура системного ПЗ відеосервера на базі ESP32	1		
3		КІ 22009.22.03.03 ПЗ	Архітектура стека	1		
			протоколів передавання			
			відеоданих			
4		КІ 22009.22.03.03 ПЗ	Схема електрична	1		
			принципова програмно-			
			технічного засобу			

					КІ 22009.22.03.03 ПЗ			
Зм	Арк	№ докум	Підпис	Дата				
Розробив		Височинський			Програмно-технічний засіб вбудованої відеосистеми з ESP32 та OV7670 із передаванням даних через WebSocket	Літера	Аркуш	Аркушів
Перевір.		Лисенко				1	1	57
						ХНУ КІ2-22-3		
Н. контр.		Лисенко						
Затв.		Павлова						

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-технічний засіб вбудованої відеосистеми з ESP32 та OV7670 із передаванням даних через WebSocket».

Автор роботи: Владислав ВИСОЧИНСЬКИЙ.

Керівник роботи: Сергій ЛИСЕНКО.

Пояснювальна записка: 57с., 14 рис., 8 табл., 5 дод., 50 джерел.

Графічна частина: 3 креслення.

ВБУДОВАНА СИСТЕМА, ВІДЕОСТРІМІНГ, ДИСТАНЦІЙНИЙ МОНІТОРИНГ, КОМП'ЮТЕРНА ІНЖЕНЕРІЯ, МІКРОКОНТРОЛЕР, РЕАЛЬНИЙ ЧАС, ESP32, OV7670, WEBSOCKET.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню програмно-технічного засобу вбудованої відеосистеми на базі мікроконтролерної платформи ESP32 та оптичного сенсора OV7670. Актуальність теми зумовлена стрімким розвитком концепції Інтернет речей (IoT) та зростаючим попитом на компактні, автономні та енергоефективні рішення для візуального контролю, автоматизації та безпеки.

Метою роботи є проєктування, реалізація та тестування апаратно-програмного комплексу для захоплення, обробки та бездротового передавання відеопотоку в реальному часі. Для досягнення поставленої мети було проведено аналіз архітектурних особливостей сучасних систем на кристалі, обґрунтовано вибір протоколу WebSocket для забезпечення низької латентності зв'язку та використання механізмів прямого доступу до пам'яті (DMA) для розвантаження процесора. Розроблено структурну та функціональну схеми пристрою, спроєктовано системне програмне забезпечення мікроконтролера з ефективним розподілом задач між обчислювальними ядрами, а також створено клієнтський вебінтерфейс для візуалізації потоку в браузерному середовищі.

_____ 30.05.2026

Підпис здобувача Дата

ЗМІСТ

Вступ 8
1. Аналіз предметної Області та методів побудови вбудованих відеосистем 9
1.1 Аналіз архітектурних особливостей сучасних систем відеоспостереження на базі мікроконтролерів 9
1.2 Аналіз мережевих протоколів передавання мультимедійних даних у системах реального часу 11
1.3 Огляд наявних рішень та аналіз їхніх недоліків 14
1.4 Тенденції розвитку ринку вбудованих відеосистем та сфери їх застосування 16
1.5 Завдання на проєктування 19
1.6 Висновок до першого розділу 20
2. Обґрунтування вибору та технічний аналіз компонентів системи 23
2.1 Обґрунтування вибору та аналіз архітектури мікроконтролера ESP32 23
2.2 Технічний аналіз та режими роботи відеосенсора OV7670 27
2.3 Архітектура та принцип формування зображення 28
2.4 Інтерфейс передачі даних DVP та часові діаграми 31
2.5 Аналіз інтерфейсу передачі даних DVP та часових діаграм синхронізації 34
2.6 Обґрунтування вибору програмного стека та аналіз операційної системи реального часу 38
2.7 Технічний аналіз структури кадру та механізмів обміну даними за протоколом WebSocket 41

2.8 Аналіз схемотехнічних рішень та забезпечення електромагнітної сумісності 43
2.9 Висновок до другого розділу 45
3. Проєктування та практична реалізація 48
3.1 Розробка схеми електричної принципової та апаратна інтеграція 48
3.2 Системна логіка та алгоритми функціонування ПЗ 50
3.3 Реалізація системного ПЗ та клієнтської частини на базі WebSocket ... 52
3.4. Підготовка середовища розробки та структура проєкту..... 54
3.5 Налаштування системних залежностей та програмного стека 55
3.6 Організація вебсервера для розгортання клієнтського інтерфейсу 56
3.7 Висновок до третього розділу 60
Висновок 63
Перелік використаних джерел 66
Додаток А Копія креслення «Загальна архітектура системи» 71
Додаток Б Копія креслення «Алгоритм функціонування системного» 72
Додаток В Копія креслення «Схема електрична принципова підключення» 73
Додаток Г код «index.html» 74
Додаток Д код «mock_esp32.py» 75

ВСТУП

Сучасні системи відеоспостереження на базі вбудованих платформ дедалі активніше застосовуються в задачах моніторингу, автоматизації та аналізу подій у режимі реального часу. Поєднання мікроконтролера ESP32 із сенсором OV7670 утворює доступну та енергоефективну апаратну основу, придатну як для локального, так і для віддаленого спостереження. Протокол WebSocket забезпечує повнодуплексний зв’язок між пристроєм і клієнтськими застосунками, що є ключовою умовою для стрімінгу відео, передачі метаданих і дистанційного керування системою.

Мета роботи - розробити та дослідити прототип вбудованої відеосистеми, що охоплює апаратну інтеграцію ESP32 з камерою OV7670, реалізацію програмного забезпечення для захоплення й попередньої обробки відеопотоку, а також організацію надійного каналу передачі даних через WebSocket. Окремо планується оцінити продуктивність системи: затримки при передачі кадрів, якість зображення після стиснення та стійкість зв'язку за різних мережевих умов.

Для досягнення цієї мети розв’язуються такі завдання: аналіз технічних характеристик ESP32 та OV7670, вибір методів кодування і стиснення кадрів, розробка прошивки з реалізацією WebSocket-клієнта і сервера, створення веб-інтерфейсу для прийому та відображення відеопотоку, а також проведення експериментів для вимірювання затримки, пропускної здатності й енергоспоживання. Особливу увагу приділено оптимізації обробки зображень в умовах обмежених ресурсів мікроконтролера та забезпеченню ефективної взаємодії між апаратною і програмною складовими системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ПОБУДОВИ ВБУДОВАНИХ ВІДЕОСИСТЕМ

1.1 Аналіз архітектурних особливостей сучасних систем відеоспостереження на базі мікроконтролерів

Стрімкий розвиток напівпровідникової промисловості та вдосконалення технологій виготовлення інтегральних схем відкрили шлях до появи нового класу мікроконтролерів із суттєво зрослою обчислювальною потужністю. Сучасні представники цього класу здатні виконувати задачі цифрової обробки сигналів, які ще кілька років тому були доступні лише спеціалізованим сигнальним процесорам або повноцінним персональним комп'ютерам [1-6]. Вбудовані відеосистеми – це клас апаратно-програмних комплексів, у яких захоплення, первинна обробка та передавання відеоінформації відбуваються безпосередньо на кінцевому пристрої з обмеженими обчислювальними ресурсами. На відміну від класичних рішень на базі архітектури x86 або потужних одноплатних комп’ютерів із повноцінними операційними системами, такі системи працюють в умовах жорстких обмежень: малого обсягу оперативної пам'яті, низького енергоспоживання та суворої детермінованості всіх процесів [12-15].

З погляду комп'ютерної інженерії, головна складність при проєктуванні таких систем полягає в організації ефективного конвеєра даних між оптичним сенсором, оперативною пам’яттю та комунікаційним інтерфейсом. Відеопотік за своєю природою генерує великий обсяг даних за короткий проміжок часу, через що процесор мікроконтролера ризикує бути повністю зайнятим переміщенням байтів - і тоді на підтримку мережевого стека чи обробку команд керування просто не залишається ресурсів [16-28]. Саме тому сучасні системи на кристалі, орієнтовані на роботу з мультимедіа, обов’язково містять контролери прямого доступу до пам’яті. Такий підхід дозволяє розвантажити обчислювальне ядро:

задачу переміщення даних від периферійних регістрів інтерфейсу камери до буфера в оперативній пам’яті бере на себе окремий апаратний блок.

Завдяки цьому система здатна підтримувати стабільну частоту кадрів і утримувати затримки на мінімальному рівні.

Окремої уваги при аналізі апаратної частини заслуговують інтерфейси взаємодії між сенсором і мікроконтролером [31-34]. У бюджетному сегменті вбудованих систем широко застосовуються паралельні інтерфейси передавання даних, зокрема DVP. Він передбачає окремі лінії для піксельних даних - зазвичай 8 біт паралельно - а також лінії синхронізації кадрів, рядків і тактового сигналу пікселів. Така організація шини вимагає від мікроконтролера достатньої кількості вільних портів введення-виведення та або високої швидкості їх опитування, або апаратної підтримки паралельного захоплення [34-39]. Альтернативою слугують послідовні інтерфейси типу MIPI CSI, які забезпечують значно вищу пропускну здатність. Втім, за це доводиться платити складнішою схемотехнікою та необхідністю спеціалізованих контролерів, які рідко трапляються в мікроконтролерах середнього цінового діапазону [31-44]. Отже, для економічно доступних рішень паралельна шина даних залишається фактичним стандартом - попри чутливість до електромагнітних завад і обмеження на довжину провідників [41-43].

Не менш важливим аспектом проєктування є вибір типу пам’яті. Вбудована статична оперативна пам’ять мікроконтролерів вирізняється найвищою швидкодією, однак її обсяг зазвичай вимірюється сотнями кілобайт - цього не вистачає навіть для зберігання одного повного кадру у високій роздільній здатності без стиснення. Розробники змушені обирати один із двох шляхів: підключати зовнішню динамічну пам’ять (PSRAM) через послідовні інтерфейси SPI або QSPI, миряючись із більшими затримками доступу, або реалізовувати алгоритми рядкової обробки й передавання даних безпосередньо під час захоплення - без повної буферизації кадру. Другий підхід потребує точної синхронізації між процесами захоплення та мережевої передачі, інакше

внутрішні буфери переповнюються і частина зображення безповоротно втрачається. Нижче на рисунку 1.1 показано класифікацію вбудованих відеосистем за призначеням.

Рисунок 1.1 – Класифікація вбудованих відеосистем за призначенням

1.2 Аналіз мережевих протоколів передавання мультимедійних даних у системах реального часу

Ефективність вбудованої відеосистеми визначається не лише швидкістю захоплення зображення, а й здатністю вчасно доставити його споживачеві через канали зв’язку. У бездротових мережах Wi-Fi, які є основним середовищем передавання для мобільних та автономних пристроїв, вибір протоколу обміну даними безпосередньо впливає на затримку відео та стабільність з’єднання.

Традиційні підходи на основі HTTP у задачах потокового відеоспостереження мають низку характерних обмежень [43-44].

Метод HTTP-запитів, який часто застосовується для отримання статичних зображень або потоків типу MJPEG, працює за принципом «запит-відповідь»: клієнт звертається до сервера, сервер повертає кадр і або закриває з'єднання, або утримує його відкритим для наступної порції даних. Головна проблема такого підходу - значні накладні витрати: кожен запит супроводжується HTTP-заголовками з метаданими, які не несуть жодної корисної інформації про зображення, але споживають смугу пропускання. До того ж встановлення нового TCP-з'єднання для кожного кадру додатково навантажує мережевий стек мікроконтролера, і без того обмеженого в ресурсах. Усе це неминуче веде до зростання затримок і падіння ефективної частоти кадрів.

Протокол RTSP у поєднанні з RTP є визнаним стандартом для професійних систем відеоспостереження - і не випадково, адже він розроблявся саме для керування мультимедійними потоками. Однак повноцінна реалізація RTSP-сервера на мікроконтролері з обмеженими ресурсами - завдання нетривіальне. Цей стек потребує підтримки складних механізмів узгодження параметрів сесії та, як правило, використовує UDP для передавання даних, що не гарантує доставку пакетів. Втрата окремих пакетів у відеопотоці може бути цілком прийнятною, проте в мережах із високим рівнем завад це обертається помітними артефактами зображення [44-46].

Для вбудованих систем, що працюють у браузерному середовищі, найперспективнішим рішенням є технологія WebSocket. Цей протокол встановлює постійне повнодуплексне з'єднання поверх одного TCP-сокету: після етапу рукостискання канал переходить у бінарний режим і залишається відкритим весь час роботи. Кадри відеопотоку передаються як бінарні масиви з мінімальним службовим навантаженням - заголовок WebSocket-кадру займає лише кілька байтів. Оскільки з'єднання не розривається між кадрами, стек TCP/IP мікроконтролера суттєво розвантажується, а затримка передавання

наближається до рівня реального часу. Двосторонній характер протоколу дає змогу організувати канал зворотного зв'язку для команд керування камерою - без жодних додаткових серверів чи портів [46-50].

З погляду комп’ютерної інженерії, ключова перевага WebSocket полягає в переході від текстового до бінарного обміну даними після завершення фази рукостискання. Якщо в HTTP кожен кадр супроводжується надлишковими заголовками, то WebSocket-фрейм обходиться мінімальним заголовком від 2 до 14 байтів - це дає змогу передавати JPEG-кадри з камери OV7670 практично без витрат пропускної здатності на службову інформацію. Для систем на базі ESP32 це має особливе значення: вивільнені ресурси процесора можна спрямувати на роботу з DMA-контролером і Wi-Fi стеком. Нижче наведено таблицю 1.1 в якій порівнюються протоколів передавання відеопотоку.

Таблиця 1.1 – Порівняльний аналіз протоколів передавання відеопотоку

Характеристика	HTTP(MJPEG)	RTSP/RTP	WebSocket
Тип з’єднання	Запит-відповідь	Потокове	Постійне двостороннє
Транспортний рівень	TCP	UDP/TCP	TCP
Затримка (Latency)	Висока	Низька	Мінімальна
Службове навантаження	Високе	Середнє	Дуже низьке
Складність реалізації	Низька	Дуже висока	Середня
Підтримка браузера	Повна	Через проксі	Повна(Native)

Отже поєднання WebSocket з ефективним апаратним забезпеченням мікроконтролера дає можливість побудувати збалансовану систему у якій накладні витрати на комунікацію зведено до мінімуму а обчислювальні ресурси повністю зосереджені на роботі з відеоданими. Для пристроїв на базі сучасних систем на кристалі такий підхід є оптимальним балансом між складністю реалізації та досягнутою продуктивністю.

1.3 Огляд наявних рішень та аналіз їхніх недоліків

У сфері відеонагляду та вбудованої електроніки представлено широкий спектр пристроїв, які розрізняються за архітектурою обчислювального ядра та рівнем програмної абстракції. У промисловому сегменті найпоширенішими є мережеві відеокамери на базі спеціалізованих систем на кристалі з апаратними засобами кодування відеосигналу. Такі пристрої, як правило, працюють під керуванням вбудованих операційних систем на основі ядра Linux. Вони забезпечують високу якість зображення та ефективне стиснення потоку, однак для автономних пристроїв із батарейним живленням підходять погано. Потужні обчислювальні ядра й блоки кодування споживають надто багато енергії для тривалої роботи від акумулятора. До того ж закрите програмне забезпечення виробника ускладнює інтеграцію таких камер у нестандартні системи керування, де потрібен прямий доступ до регістрів налаштування оптичного перетворювача.

Вбудований Linux формує значний рівень програмної абстракції, який у певних сценаріях стає обмеженням. Зокрема, доступ до сенсора OV7670 у таких системах зазвичай відбувається через стандартні драйвери ядра V4L2, що вносить додаткову затримку через контекстні перемикання між простором користувача та простором ядра. Для задач із реакцією в межах мілісекунд - наприклад, у контурах керування роботами - така затримка є неприйнятною.

Окремий клас становлять одноплатні обчислювальні модулі загального призначення, достатньо потужні для програмної обробки відеопотоку. Вони

відкривають можливості для реалізації складних алгоритмів комп'ютерного зору, проте для простого потокового передавання відео це рішення є очевидно надлишковим [49-50].

Операційна система загального призначення вносить недетерміновані затримки в обробку переривань і передавання даних через мережевий стек - що є неприйнятним для систем жорсткого реального часу. Вартість і габарити таких рішень також нерідко виходять за допустимі межі для компактних вбудованих пристроїв, призначених для мобільних платформ або побутової автоматизації.

Для систематизації результатів огляду виконано порівняльний аналіз розглянутих засобів відеофіксації за ключовими параметрами комп'ютерної інженерії - результати зведено в таблиці 1.2.

Таблиця 1.2 – Порівняння апаратних платформ для побудови відеосистем

Платформа	Енергоспоживання	Детермінованість (Real-time)	Гнучкість налаштувань	Вартість
IP-камера (Linux)	Висока	Низьке	Обмежена	Середня
SBC (Raspberry Pi)	Дуже високе	Низьке (через ОС)	Висока	Висока
8/16-біт МК (AVR)	Дуже низьке	Високе	Повна	Дуже низька
ESP32	Низьке	Високе (FreeRTOS)	Повна	Низька

У сегменті мікроконтролерів існують рішення на базі 8- та 16-розрядних архітектур, однак їхня розрядність і тактова частота є недостатніми для обробки великих масивів відеоданих. Обмежена пропускна здатність шини не дозволяє забезпечити прийнятну частоту оновлення кадрів - відеопотік фактично перетворюється на послідовність статичних зображень із помітними часовими

паузами. Спроби реалізувати передавання відео на таких платформах зазвичай зводяться до застосування зовнішніх послідовних буферів, що ускладнює схемотехніку й здорожує кінцевий виріб.

Найближчими до поставлених вимог є рішення на базі 32-розрядних мікроконтролерів із вбудованими модулями бездротового зв'язку. Наявні програмні реалізації для таких платформ здебільшого спираються на HTTP-передавання потоку у форматі MJPEG. Головний недолік цього підходу - блокувальний характер мережевих операцій і відсутність оптимізації роботи з пам'яттю. Передаючи кожен кадр як окремий HTTP-запит, мікроконтролер витрачає значну частину процесорного часу на обробку заголовків і керування з’єднанням, що неминуче знижує частоту кадрів і збільшує затримку.

Більшість відкритих бібліотек не використовують можливості DMA повною мірою та виконують копіювання даних через проміжні буфери програмним шляхом - це і є вузьке місце всієї системи. Отже, існує реальна потреба у розробці спеціалізованого програмно-технічного засобу, який усуне ці архітектурні недоліки через використання низькорівневих механізмів керування периферією та ефективніших протоколів обміну даними.

1.4 Тенденції розвитку ринку вбудованих відеосистем та сфери їх застосування

Стрімкий розвиток концепції Інтернет речей (IoT) кардинально змінив підхід до проєктування систем дистанційного спостереження та моніторингу. Якщо ще десятиліття тому передавання відео в реальному часі вимагало потужних серверних обчислювальних платформ та виділених каналів зв'язку, то сьогодні ці задачі вирішуються на рівні компактних мікроконтролерних пристроїв із бездротовим підключенням. За даними аналітичних досліджень, кількість підключених IoT-пристроїв у світі перевищила 15 мільярдів і продовжує зростати, при цьому значну частку становлять системи з функціями

захоплення та передавання зображень. Саме цей контекст визначає актуальність розробки ефективних і доступних програмно-технічних засобів вбудованого відеострімінгу.

Промислова автоматизація є одним із найбільш вимогливих секторів застосування вбудованих відеосистем. У виробничих середовищах камери на базі мікроконтролерів використовуються для контролю якості продукції на конвеєрі, моніторингу стану технологічного обладнання та виявлення аномалій у режимі реального часу. Ключовою вимогою тут є детермінована затримка передавання кадру - відхилення навіть у кілька десятків мілісекунд може спричинити помилкові спрацювання системи керування або пропуск дефектного виробу. Класичні рішення на базі промислових камер із підключенням до потужного сервера не завжди є прийнятними через вартість інфраструктури та складність розгортання в умовах розосереджених виробничих ліній, тоді як вбудовані платформи на кшталт ESP32 здатні виконувати первинну обробку прямо на місці збору даних.

Сфера розумного будинку та побутової автоматизації формує попит на принципово інший клас пристроїв - компактних, енергоефективних і простих у налаштуванні. Системи внутрішнього відеоспостереження, відеодомофони, камери для моніторингу дітей або домашніх тварин стали масовим споживчим продуктом. Для цього сегменту критично важливим є поєднання низького енергоспоживання, можливості живлення від батареї або невеликого акумулятора та підтримки хмарних і локальних протоколів зв'язку. Традиційні IP-камери на базі Linux-систем споживають надто багато енергії для автономної роботи, тоді як мікроконтролерні платформи із вбудованим Wi-Fi-модулем забезпечують повноцінний відеострімінг при споживанні на порядок меншому.

Охоронні та наглядові системи висувають вимоги до надійності каналу передавання і мінімальної залежності від централізованої інфраструктури. Пристрої периметральної безпеки, відеофіксатори на транспортних засобах та мобільні спостережні платформи повинні функціонувати навіть за нестабільного

мережевого з'єднання, підтримуючи локальний буфер даних і відновлюючи передавання після відновлення зв'язку. Тут архітектура на базі ESP32 із протоколом WebSocket демонструє очевидну перевагу: постійне з'єднання не потребує повторного рукостискання для кожного кадру, а вбудовані механізми ping/pong дозволяють автоматично виявляти розрив і ініціювати повторне підключення.

Робототехніка і безпілотні системи утворюють окремий напрямок, де вбудоване відеобачення є невід'ємною складовою навігації та взаємодії із середовищем. Мобільні роботи, безпілотні літальні апарати та автономні транспортні засоби потребують не просто передавання зображення, а замкнутого контуру зворотного зв'язку між відеосенсором і системою керування. Двосторонній характер протоколу WebSocket дозволяє реалізувати цей контур у рамках єдиного постійного з'єднання: відеопотік іде від пристрою до оператора, а команди керування орієнтацією камери або параметрами зйомки - у зворотному напрямку без жодних додаткових серверних компонентів.

Медицина і телемедицина є ще одним перспективним напрямком, де мініатюрні відеосистеми застосовуються для дистанційного моніторингу пацієнтів, передавання зображень діагностичного обладнання та забезпечення відеозв'язку між лікарем і пацієнтом у важкодоступних районах. Тут особливого значення набуває якість зображення при обмеженій пропускній здатності каналу, що робить вибір ефективного протоколу передавання та методу кодування кадрів принципово важливим проєктним рішенням.

Спільною тенденцією для всіх перелічених сфер є перехід від централізованих архітектур, де обробка ведеться виключно на сервері, до розподілених систем із первинною обробкою даних безпосередньо на пристрої - концепція, відома як Edge Computing. Вбудована відеосистема на базі ESP32 є типовим представником цього підходу: мікроконтролер самостійно захоплює кадр, виконує початкову обробку та формує стиснений потік для передавання, знімаючи навантаження з хмарної інфраструктури. Це не лише знижує вимоги

до пропускної здатності мережі, а й підвищує відмовостійкість системи в цілому, оскільки пристрій залишається функціональним навіть при тимчасовій недоступності хмарного сервера.

Таким чином, актуальність розробки програмно-технічного засобу вбудованої відеосистеми визначається збіжністю декількох факторів: зростаючим попитом на компактні автономні пристрої спостереження в різних галузях, доступністю мікроконтролерних платформ із достатньою обчислювальною потужністю для обробки відео, а також відсутністю відкритих оптимізованих реалізацій, що повною мірою використовують апаратні можливості сучасних систем на кристалі для мінімізації затримки та енергоспоживання.

1.5 Завдання на проєктування

Метою роботи є розробка та реалізація програмно-технічного засобу вбудованої відеосистеми, що забезпечує захоплення, обробку та бездротове передавання відеопотоку в реальному часі з мінімальною затримкою. Об’єктом проєктування є апаратно-програмний комплекс на базі двоядерного мікроконтролера та оптичного датчика зображення. Предметом розробки - методи та алгоритми взаємодії програмного забезпечення з апаратними ресурсами мікроконтролера для ефективного потокового передавання даних через канал із постійним з’єднанням.

Для досягнення поставленої мети необхідно розв’язати низку інженерних завдань. Насамперед потрібно спроєктувати схему підключення оптичного датчика до мікроконтролера з урахуванням обмеженої кількості вільних виводів і вимог до високошвидкісних ліній передавання даних, забезпечити стабільну генерацію тактового сигналу та коректну роботу послідовного інтерфейсу керування для конфігурації параметрів зображення. Окремої уваги потребують

узгодження рівнів напруги та мінімізація електромагнітних завад на лініях паралельної шини.

На рівні системного програмування необхідно розробити модуль керування, що використовує апаратні інтерфейси в режимі паралельного захоплення даних. Ключовим тут є налаштування DMA-контролера для автоматичного переміщення даних від периферії до оперативної пам'яті без участі процесора - це вивільняє обчислювальні ресурси для роботи мережевого стека і підтримки стабільності з’єднання.

Наступним завданням була реалізація WebSocket-сервера з підтримкою асинхронної роботи з кількома клієнтами. Необхідно розробити механізм формування пакетів відеоданих, що дозволить передавати кадри частинами або цілком залежно від розміру буферів і MTU мережі. Програмне забезпечення має працювати під керуванням операційної системи реального часу, що вимагає правильного розподілу завдань за пріоритетами та ядрами: одне завдання відповідає за захоплення кадрів і обробку сигналів синхронізації, інше - за передавання даних у мережу.

Окремою вимогою є керування параметрами відеосистеми - роздільною здатністю, якістю, налаштуваннями датчика - через зворотний канал зв’язку, що передбачає розробку протоколу обміну командами та їх інтерпретацію на стороні мікроконтролера. Результатом роботи має стати діючий макет пристрою разом із пакетом програмного забезпечення, що демонструє стабільне передавання відеопотоку з частотою кадрів, достатньою для візуального моніторингу, і затримкою в межах, прийнятних для інтерактивного керування.

1.6 Висновок до першого розділу

У першому розділі проведено комплексний аналіз предметної області, що охоплює архітектурні особливості сучасних вбудованих відеосистем, методи передавання мультимедійних даних у реальному часі та наявні апаратно-програмні рішення.

Встановлено, що вбудовані відеосистеми функціонують в умовах жорстких обмежень обчислювальних ресурсів, тому ефективна організація конвеєра даних між оптичним сенсором, оперативною пам'яттю та комунікаційним інтерфейсом є центральною інженерною проблемою при їх проєктуванні. Показано, що використання апаратних контролерів прямого доступу до пам'яті є необхідною умовою для звільнення обчислювального ядра від задач переміщення даних та забезпечення стабільної частоти кадрів. Аналіз інтерфейсів взаємодії між сенсором та мікроконтролером виявив, що паралельна шина DVP залишається фактичним стандартом для бюджетних вбудованих рішень попри її чутливість до електромагнітних завад.

Порівняльний аналіз мережевих протоколів показав, що підхід на основі HTTP-запитів та MJPEG-потоку породжує надмірні накладні витрати через службові заголовки та нові TCP-з'єднання для кожного кадру. Протокол RTSP у поєднанні з RTP, хоча й є галузевим стандартом, потребує складної програмної реалізації та не гарантує доставку пакетів під час передавання по UDP. Натомість протокол WebSocket із його постійним повнодуплексним з'єднанням і мінімальним службовим навантаженням визначено як оптимальний для вбудованих систем: він суттєво знижує затримку передавання та дає змогу організувати канал зворотного зв'язку для дистанційного керування без додаткових портів і серверів.

Огляд наявних рішень засвідчив, що промислові IP-камери на базі Linux, попри якість зображення, мають надто високе енергоспоживання й закриту архітектуру, одноплатні комп'ютери загального призначення є надлишковими для задач простого відеострімінгу, а 8- та 16-розрядні мікроконтролери не забезпечують необхідної пропускної здатності шини. Більшість відкритих реалізацій для 32-розрядних платформ не задіюють DMA повною мірою, виконуючи копіювання даних програмним шляхом, що є вузьким місцем усієї системи.

За результатами аналізу визначено мету і сформульовано технічне завдання на проєктування: розробити програмно-технічний засіб вбудованої відеосистеми на базі мікроконтролера ESP32 та оптичного сенсора OV7670 із передаванням відеопотоку через протокол WebSocket. Заплановано забезпечити низькорівневу взаємодію з апаратними ресурсами мікроконтролера через механізми DMA, реалізувати WebSocket-сервер з асинхронною підтримкою кількох клієнтів та створити браузерний інтерфейс для візуалізації потоку, а результативність системи оцінити за показниками затримки, частоти кадрів та стабільності з'єднання.

2 ОБГРУНТУВАННЯ ВИБОРУ ТА ТЕХНІЧНИЙ АНАЛІЗ КОМПОНЕНТІВ СИСТЕМИ

2.1 Обгрунтування вибору та аналіз архітектури мікроконтролера ESP32

Ключовим при розробці програмно-технічного засобу вбудованої відеосистеми є вибір обчислювального ядра, здатного одночасно виконувати інтенсивні операції введення-виведення під час захоплення відео з OV7670 та підтримувати стек мережевих протоколів Wi-Fi і WebSocket. За результатами аналізу ринку 32-розрядних систем на кристалі для цього проєкту обрано мікроконтролер ESP32 виробництва Espressif Systems.

Основним конструктивним елементом обраної апаратної платформи, зображеної на рисунку 2.1, є інтегрований модуль ESP-WROOM-32. Він являє собою завершене рішення, що містить систему на кристалі, кварцовий резонатор на 40 МГц, флеш-пам’ять об’ємом 4 МБ для зберігання прошивки та вебресурсів, а також друковану антену для роботи в діапазоні 2.4 ГГц. Металевий екран модуля забезпечує електромагнітну сумісність і стабільність Wi-Fi передавача під час стрімінгу - що особливо важливо для запобігання перешкодам на лініях паралельної шини камери.

Рисунок 2.1 – Зовнішній вигляд плати ESP32

На ній розміщено лінійний регулятор напруги, який перетворює вхідні 5 В від USB у стабільні 3.3 В для живлення мікроконтролера й сенсора OV7670. Варто зауважити, що сенсори зображення вкрай чутливі до пульсацій живлення, тому якісна система фільтрації на платі безпосередньо впливає на рівень цифрових шумів у фінальному зображенні.

Окремої уваги заслуговує підсистема зв’язку з персональним комп’ютером. На платі розпаяно міст USB-to-UART - найчастіше CP2102 або CH340. Це дозволяє не лише завантажувати програмний код, а й вести низькорівневе налагодження через послідовний порт, відстежуючи ініціалізацію камери та стан WebSocket-з’єднання в реальному часі.

Для підключення камери OV7670 використовуються два ряди штирових роз'ємів із задіяними 16 виводами GPIO. Оскільки передавання відео здійснюється через 8-бітну паралельну шину, компактна топологія плати дає нам можливість мінімізувати довжину з'єднувальних провідників - а це безпосередньо знижує ймовірність помилок при зчитуванні даних на високих частотах.

Розгляд функціональної блок-схеми архітектури рисунок 2.2 димонструє ключові апаратні вузли, взаємодія яких забезпечує роботу вбудованої відеосистеми в режимі реального часу.

Оскільки пристрій призначений для безперервного стрімінгу, вирішальну роль відіграє не лише абсолютна продуктивність процесора, а й ефективність шинної топології та периферійних модулів.

Основу системи складають два 32-розрядні ядра Xtensa® Dual-Core LX6. Така архітектура уможливлює справжню багатозадачність на рівні операційної системи реального часу FreeRTOS. У цьому проєкті ядра розподілено таким чином:

1. Протокольне ядро (PRO_CPU) обробляє переривання від Wi-Fi модуля, підтримує стек TCP/IP та виконує логіку WebSocket-сервера.

2. Прикладне ядро (APP_CPU) повністю виділене під роботу з відеосенсором: ініціалізацію камери через інтерфейс SCCB та програмну обробку кадрів перед відправленням.

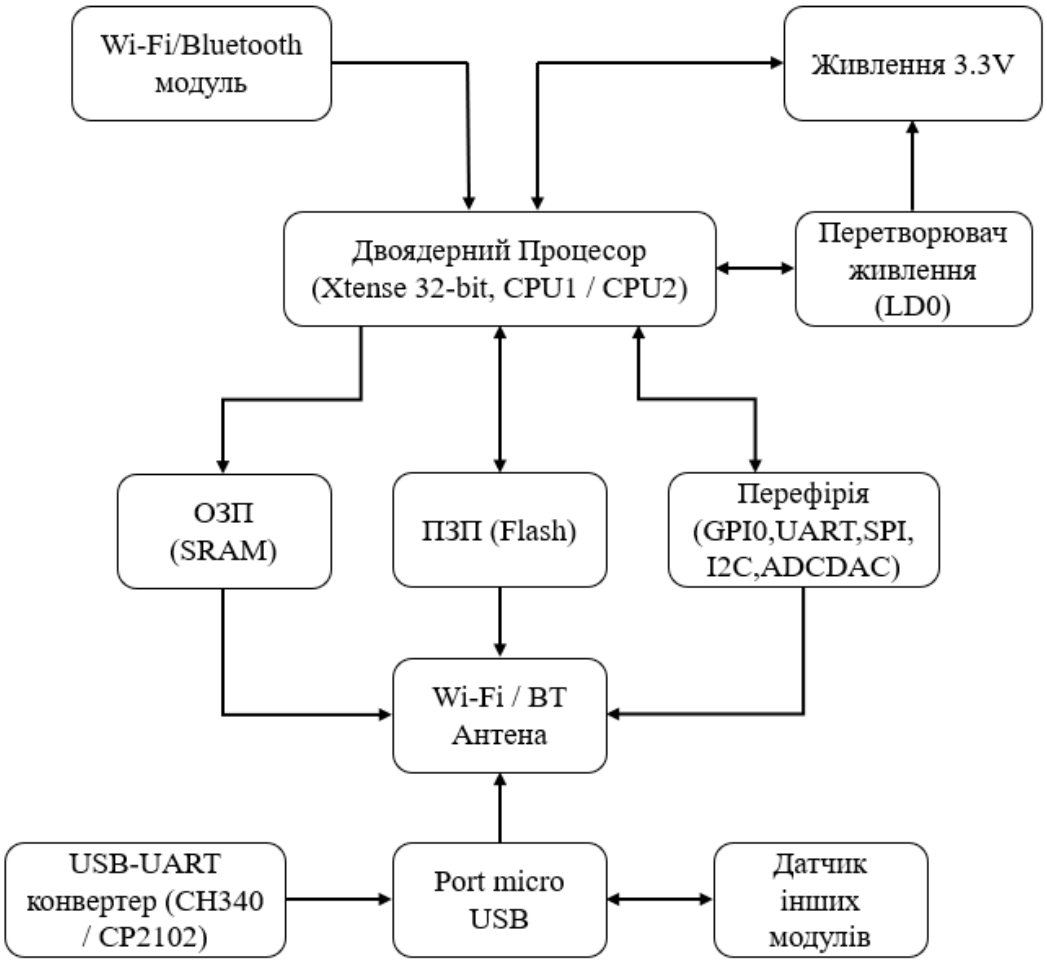

Рисунок 2.2 – Функціональна блок-схема архітектури SoC ESP32

ESP32 містить 520 КБ вбудованої SRAM - для відеосистеми цього обмаль, тому особлива роль відводиться DMA-контролеру, пов’язаному на блок-схемі з периферійними інтерфейсами. Завдяки DMA байти зображення з камери надходять у паралельний порт і автоматично записуються до буфера в оперативній пам’яті без участі процесора. Це усуває класичне «вузьке місце»: без DMA CPU витрачав би до 80% потужності на просте копіювання байтів, що унеможливило б паралельну роботу Wi-Fi стека.

Хоча модуль I2S позначений на блок-схемі як аудіоінтерфейс, у цій системі він працює в специфічному режимі Camera Mode - як 8-бітний паралельний вхід, синхронізований зовнішніми сигналами OV7670 (PCLK та HREF). Вбудовані FIFO-буфери модуля дозволяють накопичувати фрагменти рядків зображення й передавати їх у пам’ять великими блоками через шину DMA - це забезпечує стабільну частоту кадрів навіть за максимальної роздільної здатності сенсора.

Блок Wi-Fi/Bluetooth Radio містить апаратний рівень PHY, що відповідає за передавання сигналу. Для реалізації WebSocket-з’єднання важливу роль відіграють блоки апаратного прискорення шифрування - AES, SHA-2 та RSA, - також відображені на схемі архітектури. Вони дозволяють обробляти захищені пакети WSS без суттєвого навантаження на обчислювальні ядра, зберігаючи стабільний пінг і мінімальну затримку відеопотоку.

Блок Internal Oscillators & PLL забезпечує гнучке налаштування тактової частоти. Для роботи камери OV7670 мікроконтролер генерує сигнал XCLK за допомогою вбудованого генератора - LEDC або PWM. Стабільність цього сигналу, керованого безпосередньо апаратною частиною SoC, є обов’язковою умовою коректного фазового зсуву при зчитуванні піксельних даних.

Дані таблиці 2.1 підтверджують високу придатність обраної платформи для задач потокової обробки відео. Тактова частота 240 МГц забезпечує достатню обчислювальну потужність для програмного стиснення кадрів у формат JPEG - що є необхідністю при роботі з сенсором OV7670, який не має вбудованого JPEG-кодера.

Значну роль відіграють і 520 КБ вбудованої SRAM. Один нестиснений кадр у роздільній здатності QVGA (320×240) у форматі RGB565 займає близько 153.6 КБ, тож мікроконтролер здатний одночасно утримувати в пам’яті буфер для захоплення нового кадру та буфер для передавання попереднього через WebSocket - тобто повноцінно реалізувати метод подвійної буферизації.

Таблиця 2.1 - Основні технічні характеристики мікроконтролера ESP32

Параметри	Значення
Обчислювальне ядро	Xtensa® Dual-Core 32-bit LX6
Максимальна тактова частота	240 МГц
Продуктивність	600 DMIPS
Оперативна пам’ять (SRAM)	520 КБ
Підтримка зовнішньої пам’яті	до 16 МБ (Flash), до 8 МБ (PSRAM)
Стандарт бездротового зв’язку Wi-Fi	IEEE 802.11 b/g/n (до 150 Мбіт/с)
Версія Bluetooth	v4.2 BR/EDR та BLE
Переферійні інтерфейси	UART, SPI, I2C, I2S, CAN, ADC, DAC
Апаратне прискориня криптографії	AES, SHA-2, RSA, ECC, RNG
Робоча напруга живлення	2.2 В - 3.6 В
Кількість виводів загального призначення (GPIO)	34

Окремо можна відзначити апаратний блок I2S. Попри те що цей інтерфейс традиційно використовується для аудіоданих, у цьому пристрої він конфігурується для паралельного захоплення 8-бітних даних від камери. Це відкриває можливість використання DMA, що знижує навантаження на CPU під час операцій введення-виведення практично до нуля й вивільняє ресурси для роботи мережевого стека Wi-Fi.

2.2 Технічний аналіз та режими роботи відеосенсора OV7670

Вибір пристрою захоплення зображення є одним з ключових етапів проєктування вбудованої відеосистеми. Саме апаратні можливості сенсора визначають якість первинних даних, навантаження на обчислювальне ядро мікроконтролера та потенційну пропускну здатність каналу зв’язку. Для

реалізації цього програмно-технічного засобу обрано CMOS-матрицю OV7670 виробництва OmniVision Technologies.

Вибір обгрунтовано балансом між вартістю, енергоспоживанням і функціональною гнучкістю. OV7670 являє собою систему на кристалі для обробки зображень, що інтегрує не лише світлочутливу матрицю, а й повноцінний блок цифрової обробки сигналів.

Для роботи з мікроконтролерами серії ESP32 цей сенсор є природним вибором: він забезпечує прямий вихід цифрових даних через паралельну шину, що усуває потребу в зовнішніх аналого-цифрових перетворювачах.

Головною особливістю OV7670 є можливість повної низькорівневої конфігурації через послідовний інтерфейс керування SCCB. Це дає розробнику змогу гнучко налаштовувати параметри захоплення - від роздільної здатності та частоти кадрів до алгоритмів автоматичного підсилення й балансу білого - відповідно до умов роботи WebSocket у конкретній мережевій інфраструктурі.

На відміну від камер із вбудованим апаратним стисненням у JPEG - наприклад, OV2640 - сенсор OV7670 у базовій конфігурації видає «сирі» піксельні дані. Це ставить перед розробником задачу ефективної організації програмних буферів та алгоритмів первинної фільтрації, що є окремим предметом дослідження в цій роботі. У наступних пунктах розглянуто внутрішню архітектуру сенсора, часові діаграми його роботи та методи програмного керування режимами.

2.3 Архітектура та принцип формування зображення

Відеосенсор OV7670 базується на CMOS-архітектурі – галузевому стандарті для енергоефективних вбудованих рішень. В основу його побудови покладено концепцію «Image-on-Chip»: усі функціональні вузли, починаючи від світлочутливої матриці і закінчуючи блоками цифрової обробки та вихідними інтерфейсами, зосереджені на єдиному кристалі. Така інтеграція дає змогу

суттєво зменшити розміри модуля, придушити власні шуми сигнального тракту та знизити енергоспоживання - характеристику, що є особливо важливою при роботі у зв'язці з мікроконтролером ESP32.

Сенсор спроєктований як самодостатня система, здатна перетворювати світловий потік на цифрові дані без жодних зовнішніх допоміжних компонентів. Це стало можливим завдяки поєднанню високочутливої аналогової матриці пікселів із вбудованим конвеєром цифрової обробки сигналів (DSP). На відміну від застарілих CCD-рішень, CMOS-архітектура OV7670 підтримує довільне звернення до окремих рядків і стовпців матриці, що відкриває можливості для апаратного масштабування, панорамування та гнучкого регулювання частоти кадрів через внутрішні регістри.

Основу сенсора складає матриця активних пікселів розміром 640 * 480. Кожен піксель перетворює енергію падаючого світла на електричний заряд. Оскільки напівпровідникові сенсори самі по собі не розрізняють довжину хвилі, поверх матриці нанесено фільтр Байєра який наведений на рисунку 2.3.

У цьому пристрої застосовується стандартна мозаїка RGGB: 50% пікселів відповідають за зелений канал - відповідно до підвищеної чутливості людського ока до цього діапазону - 25% за червоний і 25% за синій. Відновлення повноколірного зображення виконує внутрішній процесор камери за допомогою алгоритмів білінійної інтерполяції.

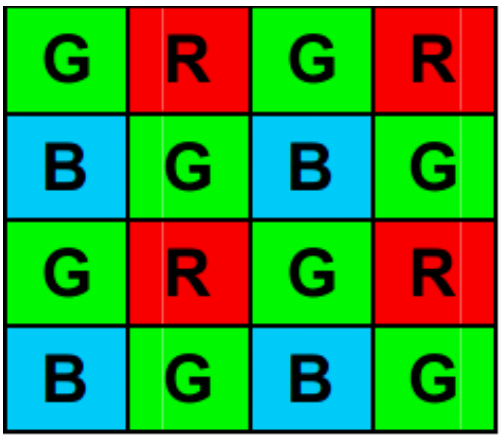

Рисунок 2.3 – Структура фільтра Байєра

Після аналого-цифрового перетворення сигнал надходить до блоку цифрової обробки DSP. Для забезпечення якісного зображення перед передаванням через WebSocket сенсор автоматично виконує низку операцій, зокрема регулює час накопичення заряду залежно від освітленості за допомогою AEC, підсилює слабкий сигнал у темряві через AGC, коригує колірну температуру згідно з умовами знімання завдяки AWB та здійснює програмне перевертання кадру за допомогою функції дзеркального відображення через регістри MVFP. Технічні параметри сенсора наведено в таблиці 2.2 технічні характеристики та параметри CMOS-матриці OV7670.

Таблиця 2.2 – Технічні характеристики та параметри CMOS-матриці OV7670

Параметр	Одиниця виміру	Значення
Розмір пікселя	мкм	3.6 x 3.6
Ефективна площа матриці	мм x мм	2.36 x 1.76
Співвідношення сигнал/шум	дБ	46
Динамічний діапазон	дБ	52
Кількість рівнів ADC	біт	10
Максимальна швидкість VGA	кадр/с	30
Споживання в режимі очікування	мкА	< 20
Кількість рівнів ADC	біт	10

Для коректного проєктування системи важливо оцінити реальний обсяг даних, який ESP32 має прийняти від камери. Нижче розраховано потік для

роздільної здатності VGA (640*480) у форматі RGB565 – 2 байти на піксель - при частоті 30 FPS за формулою 2.1 і нижче наведено результат обрахунків 2.2:

$$S=W\times H\times BPP\times FPS, \quad (2.1)$$

де: W,H – ширина та висота кадру;

BPP – кількість байтів на піксель;

FPS – частота кадрів.

$$S=640\times 480\times 2\times 30=18\ 432\ 000\ байта/с=18.4\ МБ/с. \quad (2.2)$$

2.4 Інтерфейс передавання даних DVP та часові діаграми

Для реалізації цього програмно-технічного засобу застосовано паралельний інтерфейс DVP – стандартне рішення для бюджетних вбудованих систем. На відміну від послідовних інтерфейсів типу SPI, де дані передаються побітово, паралельна шина DVP передає цілий байт за один такт. Це дає можливість досягти пропускної здатності, необхідної для стрімінгу відео, без надмірного підвищення тактової частоти - що своєю чергою знижує ризик електромагнітних завад і помилок зчитування.

Ключове завдання при проєктуванні інтерфейсу - забезпечити жорстку синхронізацію між моментом формування пікселя в матриці сенсора та моментом запису цих даних у пам’ять ESP32. Захоплення зображення є безперервним потоком, де кожен байт займає суворо визначене місце в структурі кадру. Найменше відхилення в часових діаграмах сигналів PCLK, VSYNC або HREF незворотно спотворює кадр - зміщує рядки або повністю руйнує синхронізацію.

З огляду на архітектуру ESP32, DVP розглядається не просто як набір контактів, а як повноцінний апаратний конвеєр. У наступних підпунктах

детально розглянуто кожен керуючий сигнал, розраховано часові обмеження для стабільної роботи системи та проаналізовано діаграми взаємодії сенсора з блоком I2S мікроконтролера в режимі паралельного захоплення. Ці матеріали формують необхідну основу для розробки низькорівневого драйвера, який забезпечить передавання «сирих» даних до буфера WebSocket-сервера з мінімальною затримкою.

В основі цього інтерфейсу DVP лежить 8-бітна паралельна шина даних D0-D7, по якій безпосередньо передаються цифрові значення пікселів. Проте сама по собі шина даних не може працювати без синхронізації яку забезпечують три інші сигнали. Ритм усій системі задає тактовий сигнал PCLK. Кожен його перепад слугує командою для мікроконтролера зчитати поточний байт із шини. Для того щоб приймач міг правильно з’єднати ці байти у картинку, використовуються сигнали розгортки HREF він вказує на початок і кінець кожного рядка, а VSYNC визначає межі цілих кадрів, сигналізуючи про завершення передачі поточної картинки та готовність до формування наступної.

Обов’язковою умовою роботи інтерфейсу є подача сигналу XCLK на вхід камери. OV7670 не має власного генератора, тому ESP32 формує тактову частоту в діапазоні 10-24 МГц за допомогою вбудованого модуля PWM. Стабільність внутрішніх процесів формування зображення та частота PCLK безпосередньо залежать від точності цього сигналу.

Як видно з рисунка 2.4, процес зчитування одного кадру має ієрархічну структуру, де на рівні кадру захоплення ініціалізується за фронтом сигналу VSYNC, що слугує командою для DMA-контролера ESP32 розпочати запис у новий буфер пам’яті.

На рівні рядка протягом одного циклу VSYNC сенсор генерує кількість імпульсів HREF, яка відповідає обраній вертикальній роздільній здатності (480 для VGA або 240 для QVGA), а на рівні пікселя всередині кожного такого імпульсу передаються байти даних, причому у форматі RGB565 на один піксель припадає два такти PCLK, тобто два послідовні байти.

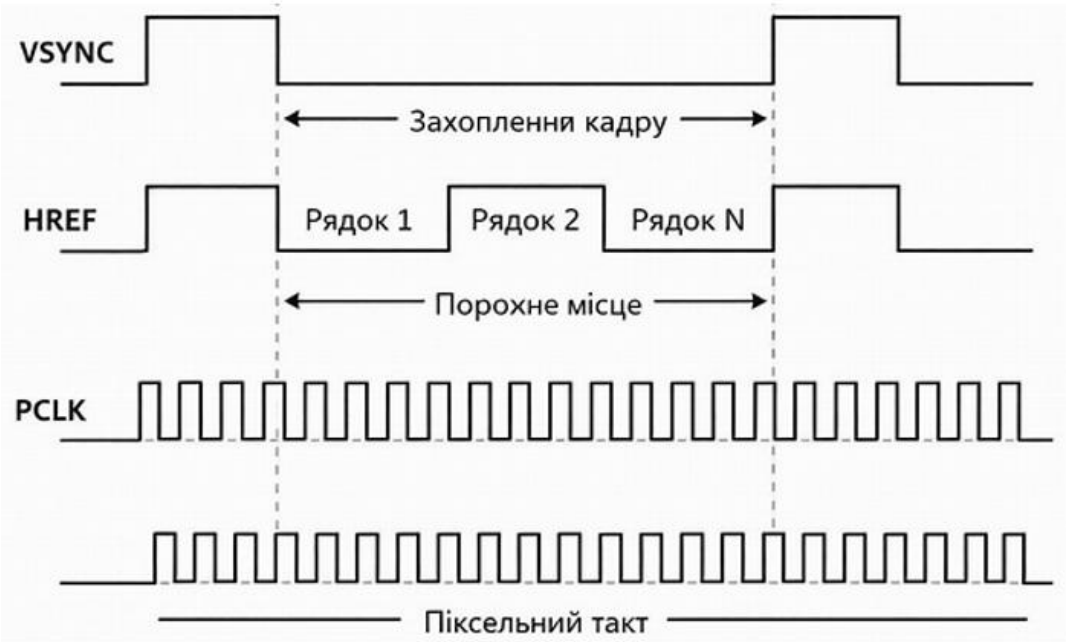

Рисунок 2.4 – Часова діаграма вихідних сигналів OV7670

Для інженерного обґрунтування пораховано необхідну частоту PCLK у режимі VGA (640*480) при 30 FPS. Оскільки кожен піксель у форматі RGB565 займає 2 байти, нижче показаний результат обрахунків 2.3 де пораховано загальну кількість байтів у кадрі і вона становить:

$$N_{bytes}=640\times480\times2=614\ 400 \text{ байт.} \quad (2.3)$$

Мінімальна частота f_{PCLK} (без урахування пауз на синхронізацію) наведена нижче 2.4:

$$f_{PCLK}\ N_{butes}\times FPS=614\ 400\times3018.4 \text{ МГц} . \quad (2.4)$$

З урахуванням службових інтервалів (blanking periods) реальна частота PCLK становить близько 24 МГц. Це робить апаратний модуль I2S обов’язковою умовою: програмне опитування портів (bit-banging) на такій частоті призведе до 100% завантаження CPU і неминучої втрати даних.

2.5 Аналіз інтерфейсу передавання даних DVP та часових діаграм синхронізації

Для реалізації цього програмно-технічного засобу застосовано паралельний інтерфейс DVP - стандартне рішення для бюджетних вбудованих систем. На відміну від послідовних інтерфейсів типу SPI, де дані передаються побітово, паралельна шина DVP передає цілий байт за один такт. Це дозволяє досягти пропускної здатності, необхідної для стрімінгу відео, без надмірного підвищення тактової частоти – що знижує ризик електромагнітних завад і помилок зчитування.

Ключове завдання при проєктуванні інтерфейсу - забезпечити жорстку синхронізацію між моментом формування пікселя в матриці сенсора та моментом запису цих даних у пам’ять ESP32. Захоплення зображення є безперервним потоком, де кожен байт займає суворо визначене місце в структурі кадру. Найменше відхилення в часових діаграмах сигналів PCLK, VSYNC або HREF незворотно спотворює кадр - зміщує рядки або повністю руйнує синхронізацію. З огляду на архітектуру ESP32, DVP розглядається не просто як набір контактів, а як повноцінний апаратний конвеєр. У наступних підпунктах детально розглянуто кожен керуючий сигнал, розраховано часові обмеження для стабільної роботи системи та проаналізовано діаграми взаємодії сенсора з блоком I2S мікроконтролера в режимі паралельного захоплення.

Якість зображення та стабільність роботи вбудованої відеосистеми, що транслює потік через WebSocket-канал, визначаються передусім точністю налаштування внутрішніх параметрів сенсора на етапі ініціалізації. У системі ESP32 + OV7670 роль центрального механізму такої взаємодії виконує протокол SCCB (Serial Camera Control Bus) - спеціалізований послідовний інтерфейс від OmniVision, що забезпечує низькорівневий доступ до конфігураційного простору камери. Саме він відповідає за первинне узгодження між мікроконтролером і сенсором, вибір схем кодування кольору та активацію

вбудованих алгоритмів обробки сигналу, перетворюючи камеру на адаптивний інструмент відео-фіксації, здатний коректно працювати за різних умов освітлення та при мережевих обмеженнях.

У рамках проєкту аналіз SCCB є особливо важливим: обмежені обчислювальні ресурси ESP32 у поєднанні зі специфікою WebSocket-протоколу вимагають максимально можливої оптимізації вхідного потоку даних ще на стадії його формування в сенсорі. Налаштування внутрішніх регістрів дає перекласти ресурсоємні обчислювальні задачі - апаратне масштабування, кадрове кадрування та субдискретизацію колірності - на внутрішні блоки камери. Це вивільняє процесорні цикли ESP32, необхідні для обслуговування стека TCP/IP, обробки асинхронних подій WebSocket-сервера та виконання криптографічних операцій у захищених мережах.

Протокол SCCB, розроблений OmniVision, функціонально сумісний із широко розповсюдженим інтерфейсом I2C і використовує дві сигнальні лінії: SIOC лінію тактування, що визначає швидкість обміну даними (зазвичай до 400 кГц) та SIOD двонаправлену лінію передавання даних.

Відмінність SCCB від стандартного I2C полягає у відсутності обов'язкового сигналу підтвердження ACK у деяких фазах передавання - це вимагає від драйвера ESP32 особливої обробки таймінгів. Запис у регістр сенсора складається з трьох послідовних фаз: передавання адреси пристрою, адреси цільового регістра та власне даних. Налаштування камери для роботи з WebSocket-сервером потребує точного встановлення значень у внутрішній пам’яті сенсора. Основні регістри, задіяні в цьому проєкті, наведено в таблиці 2.3.

Регістр COM7 за адресою 0x12 фактично виконує роль центрального диригента всієї логіки камери. Саме з нього починається будь-яка ініціалізація: перш ніж виставляти будь-які інші параметри, розумно привести пристрій до відомого стану.

Таблиця 2.3 – Основні регістри керування сенсора OV7670 та їх функціональне призначення

Адреса (Hex)	Назва регістра	Значення за замовчуванням	Опис та роль у ПТЗ
0x12	COM7	0x00	Керування форматом: скидання (bit 7), вибір VGA/QVGA/CIF, вибір RGB/YUV.
0x11	CLKRC	0x80	Внутрішній подільник частоти. Визначає швидкість PCLK відносно вхідного XCLK.
0x40	COM15	0xC0	Налаштування вихідного діапазону RGB та формату (RGB565/555).
0x0C	COM3	0x00	Керування масштабуванням (Scaling) та функцією DCW.
0x3E	COM14	0x00	Керування ручним та автоматичним масштабуванням пікселів
0x70	SCALING_XSC	0x3A	Налаштування горизонтального масштабування (для QVGA).
0x71	SCALING_YSC	0x35	Налаштування вертикального масштабування (для QVGA).
0xA2	SCALING_PCLK_DIV	0x01	Додатковий подільник для PCLK (важливо для стабільності Wi-Fi).
0x1E	MVFP	0x00	Дзеркальне відображення кадру (Mirror/Flip).
0x01	BLUE	0x80	Ручне налаштування синього каналу (Gain control).
0x02	RED	0x80	Ручне налаштування червоного каналу (Gain control).

Якщо записати до нього значення 0x80, відбудеться апаратне скидання - усі параметри повернуться до заводського стану, що усуває можливі артефакти від попередніх сесій роботи. Далі, залежно від обраного формату виводу, значення змінюється: 0x04 переводить камеру в режим RGB, а 0x00 - у режим YUV. Цей вибір не формальний і має цілком відчутні практичні наслідки: YUV422 передає яскравісну та колірну інформацію значно економніше завдяки тому, що людське око значно чутливіше до деталей яскравості, ніж до точності кольору. Саме це робить YUV природним вибором для мережевого стрімінгу, де кожен зайвий байт має значення, а надлишкова колірна інформація лише марно навантажує канал.

Окремої уваги заслуговує регістр CLKRC за адресою 0x11, який відповідає за подільники тактової частоти. На перший погляд це суто технічна деталь, однак саме тут криється одна з найпоширеніших пасток при розробці потокового відео на ESP32. Проблема прихована в архітектурі самого мікроконтролера: Wi-Fi модуль і контролер I2S ділять між собою ресурси однієї системної шини, тому якщо піксельне тактування PCLK виявиться надто високим, мережевий стек просто не встигатиме обробляти потік даних - кадри починають губитися, а з'єднання стає нестабільним. Керуючи регістром CLKRC, можна навмисно уповільнити видачу байтів камерою до такого темпу, за якого WebSocket-сервер встигає акуратно запакувати все у TCP-пакети без втрат. Це свідомий компроміс між швидкістю та надійністю, і в більшості реальних застосунків надійність перемагає.

Роздільна здатність VGA - 640×480 пікселів для потокового відео в реальному часі часто виявляється надлишковою розкішшю, яка лише марнує пропускну здатність. Через регістри COM3, COM14 та групу SCALING можна доручити самому сенсору апаратно зменшити кадр до QVGA - тобто до 320×240 пікселів. Принципово важливо розуміти, що це не програмне стиснення після факту, коли центральний процесор витрачає цикли на обробку вже захопленого зображення, а скорочення ще на рівні заліза, до того як дані взагалі потрапляють

у пам'ять. Такий підхід зменшує обсяг даних рівно вчетверо, і саме це дає нпм досягти плавного відео навіть на мобільних пристроях зі скромним каналом зв'язку, не жертвуючи при цьому частотою кадрів.

2.6 Обґрунтування вибору програмного стека та аналіз операційної системи реального часу

Розробка вбудованої відеосистеми потребує не лише надійного апаратного забезпечення, а й ефективного програмного інструментарію, здатного забезпечити детермінованість процесів. Оскільки ESP32 є багатоядерною системою, вибір середовища розробки та архітектури системного ПЗ безпосередньо впливає на стабільність відеопотоку.

Для реалізації проєкту було обрано зв'язку VS Code разом із PlatformIO – і це не просто питання зручності. Порівняно зі спрощеним Arduino IDE, такий стек дає відчутні переваги саме там, де це важливо для системного програмування.

Одна з них стосується керування залежностями. Файл platformio.ini дозволяє жорстко зафіксувати версії всіх бібліотек - і для камери, і для WebSocket - а це означає, що збірка відтворюватиметься однаково на будь-якому комп'ютері, незалежно від того, хто і коли її запускає. Жодних «у мене працювало» через оновлення сторонньої бібліотеки.

Не менш важливою є підтримка інтелектуального аналізу коду. Завдяки IntelliSense та вбудованому статичному аналізатору помилки роботи з пам'яттю - зокрема витоки відеобуферів, які в мікроконтролерних проєктах бувають особливо підступними - вдається помітити ще під час написання коду, а не під час налагодження вже зібраного пристрою.

Останьою перевагою є гнучкість на рівні компілятора. Прямий доступ до параметрів GCC дає змогу тонко підлаштувати оптимізацію під конкретну ревізію чіпа ESP32, а не покладатися на усереднені налаштування, які Arduino IDE обирає за вас.

Прошивка мікроконтролера працює під керуванням FreeRTOS – невід'ємної частини фреймворку ESP-IDF. Використання RTOS є обов'язковою умовою для цього пристрою, оскільки система має одночасно виконувати кілька критичних завдань.

У серці проєкту працює FreeRTOS, і два його механізми відіграють тут важливу роль.

Перший це витісняюча багатозадачність - планувальник завдань здатний миттєво відібрати процесор у будь-якого поточного процесу, щойно надійде переривання від камери - зокрема сигнал VSYNC. Це відбувається навіть тоді, коли в цей самий момент триває передавання Wi-Fi пакета. Саме така поведінка гарантує, що камера ніколи не чекатиме на мережевий стек, і кадри захоплюються вчасно.

Другий механізм - черги та семафори, які вирішують задачі безпечної комунікації між двома незалежними завданнями. Завдання захоплення та завдання WebSocket-сервера працюють паралельно й не можуть просто так обмінюватися даними через спільну змінну - це призвело б до гонки станів. Натомість вказівники на буфери з кадрами передаються через чергу, яка бере на себе всю синхронізацію, залишаючи кожне з завдань зосередженим на своїй роботі.

Одним із рішень у проєкті є апаратна прив'язка завдань до конкретних ядер процесора. ESP32 має два ядра - PRO_CPU та APP_CPU - і замість того щоб дозволити планувальнику довільно розподіляти навантаження між ними, кожному ядру було свідомо відведено власну зону відповідальності.

Ядро 0, PRO_CPU, повністю віддано під мережеву діяльність: підтримку Wi-Fi стека, обробку вхідних TCP-запитів і роботу WebSocket-сервера. Це означає, що все, що стосується передавання даних назовні, відбувається в ізольованому середовищі, де жодна стороння задача не може внести непередбачувану затримку.

Ядро 1, APP_CPU, натомість працює виключно з периферією камери. Саме тут виконується ініціалізація OV7670, керування DMA-транзакціями та, коли це необхідно, програмна конвертація форматів - зокрема перепакування байтів RGB. Така ізоляція надає камері працювати у власному ритмі, не конкуруючи за процесорний час із мережевим стеком, і саме це робить всю систему передбачуваною й стабільною під навантаженням.

Такий розподіл дозволяє уникнути “джиттера” (тремтіння) відеопотоку, оскільки важкі мережеві операції не заважають жорстким таймінгам зчитування пікселів. Нижче діаграма на рисунок 2.5, яка показує розподілу завдань FreeRTOS між ядрами ESP32.

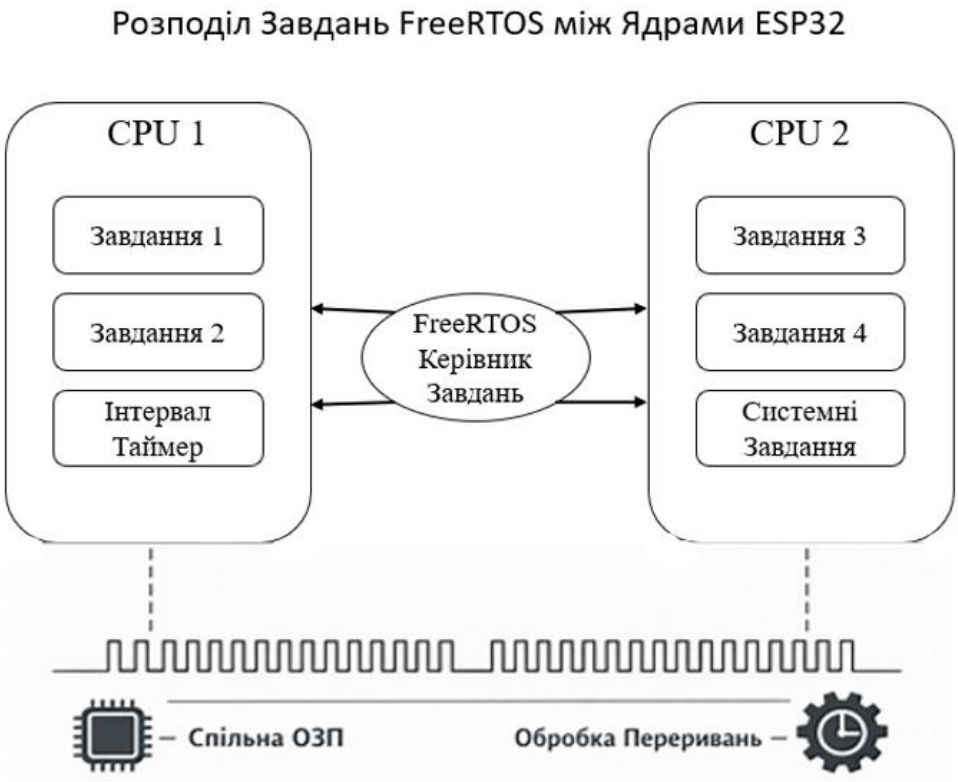

Рисунок 2.5 – Діаграма розподілу завдань FreeRTOS між ядрами ESP32

Для роботи з апаратною частиною задіяно такі спеціалізовані бібліотеки як esp32-camera це низькорівневий драйвер, що налаштовує I2S-периферію в режим паралельного захоплення, забезпечує ініціалізацію сенсора OV7670 через SCCB та конфігурацію DMA-дескрипторів та WebSocketsServer це бібліотека для

реалізації протоколу RFC 6455. Обрана завдяки підтримці асинхронних з'єднань і можливості передавання великих бінарних масивів - стандартного підходу для передавання стиснутих JPEG-зображень.

2.7 Технічний аналіз структури кадру та механізмів обміну даними за протоколом WebSocket

Для стабільного передавання відеопотоку з високою частотою кадрів необхідно мінімізувати службове навантаження на канал зв'язку. Протокол WebSocket, вирішує це завдання ефективно замість громіздких текстових заголовків HTTP він використовує компактні бінарні фрейми.

WebSocket-фрейм – це послідовність байтів, де кожен біт має чітке функціональне призначення. На відміну від HTTP, де заголовок може займати кілька кілобайт, мінімальний заголовок WebSocket становить лише 2 байти. Нижче наведена таблиця 2.4 на якій показано Бітова структура заголовка фрейму WebSocket

Таблиця 2.4 – Бітова структура заголовка фрейму WebSocket

Біти	Назва поля	Опис та значення для ПТЗ
0	FIN	Прапор завершення. Якщо 1 - це останній фрагмент повідомлення (кадру).
1-3	RSV1-3	Резервні біти. Зазвичай встановлюються в 0.
4-7	Opcode	Тип даних: 0x2 для бінарних даних (JPEG), 0x1 для тексту (команди).
8	Mask	Прапор маскування. Для даних від клієнта до сервера завжди 1, від сервера (ESP32) - 0.
9-15	Payload len	Довжина даних. Якщо <126 - вказується тут, якщо більше - використовуються додаткові байти.

Стиснутий JPEG-кадр навіть у роздільній здатності QVGA займає від 10 до 40 КБ - і це одразу створює проблему на рівні протоколу. Стандартне 7-бітне поле Payload len у заголовку WebSocket просто не розраховане на такі обсяги. Щоб обійти це обмеження, протокол передбачає розширення заголовка: якщо розмір кадру потрапляє в діапазон від 126 до 65535 байт, поле довжини виставляється у значення 126, а реальний розмір записується в наступні два байти. Для ще більших даних існує значення 127 із вісьмома додатковими байтами, хоча для мікроконтролерних застосунків такий сценарій на практиці майже не зустрічається.

Окремою проєктною позицією стало рішення щодо формату передавання і тут варто пояснити, чому обрано бінарні фрейми з Opcode 0x2, а не поширений у аматорських проєктах підхід через Base64 усередині текстових пакетів. Проблема Base64 не лише в тому, що він збільшує розмір кожного кадру на третину. Додатково ESP32 витрачає процесорні цикли на перетворення кожного байта в текстове представлення - а це безпосередньо б'є по частоті кадрів. І це ще не все: браузер на мобільному пристрої отримує таке повідомлення й змушений витрачати власні ресурси на зворотне декодування, перш ніж показати хоча б один кадр.

У цій роботі реалізовано принципово інший підхід: байти передаються напряму з DMA-буфера в сокет без жодних перетворень. Браузер отримує дані як об'єкт Blob і миттєво відображає їх на елементі Canvas через API URL.createObjectURL - без проміжних кроків і без зайвого навантаження на жодну зі сторін. Схему інкапсуляції JPEG-кадру у фрейм WebSocket проілюстровано на рисунку 2.6.

Завдяки повнодуплексній природі WebSocket реалізовано не лише передавання відео, а й канал зворотного зв'язку. Клієнт може надсилати текстові фрейми (Opcode 0x1) з командами - наприклад, `{"cmd":"bright", "val":20}`. ESP32 аналізує такий пакет у реальному часі та через інтерфейс SCCB змінює відповідні регістри OV7670, не перериваючи відеопотік.

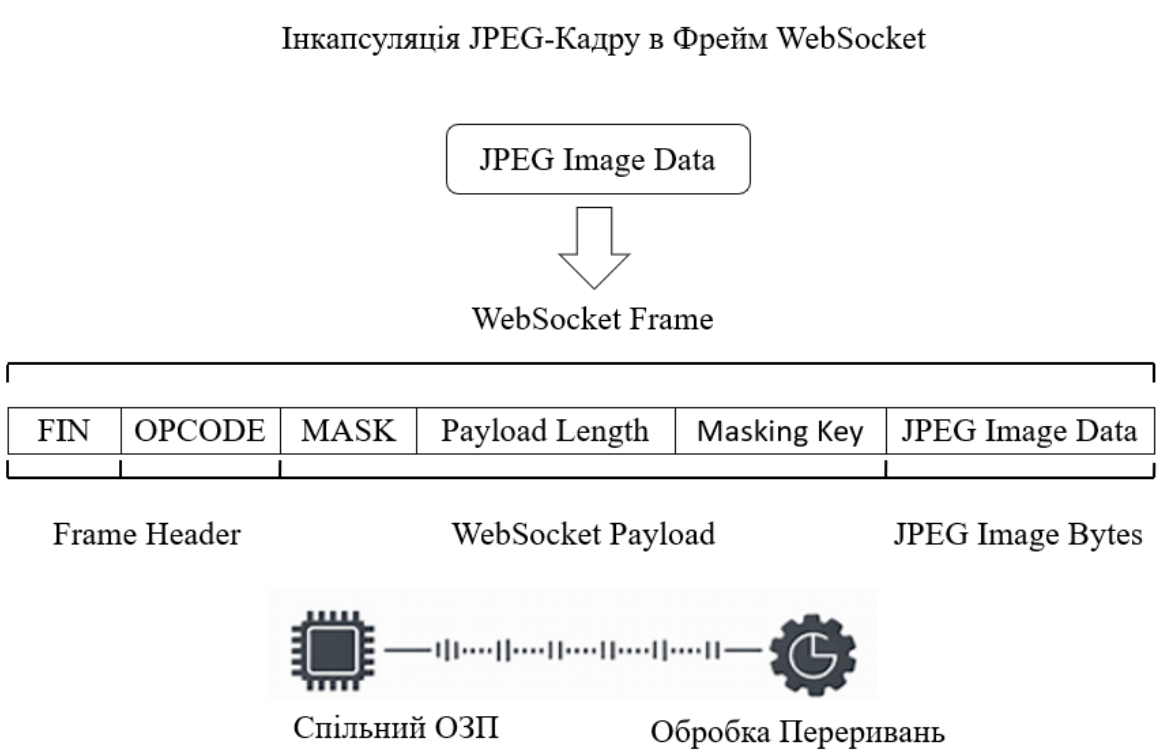

Рисунок 2.6 – Схема інкапсуляції JPEG-кадру у фрейм WebSocket

2.8 Аналіз схемотехнічних рішень та забезпечення електромагнітної сумісності

Фізична реалізація відеосистеми на базі ESP32 та OV7670 - це не лише питання правильного коду. Система працює з піксельним тактуванням до 24 МГц і водночас здійснює бездротове передавання даних, а це означає, що якість електричних з'єднань безпосередньо визначає, чи буде картинка чистою або взагалі чи буде система працювати стабільно.

Фундаментом усього є стабільність живлення. Два головні споживачі поводяться принципово по-різному, і це створює напругу вже на рівні схемотехніки. ESP32 під час роботи Wi-Fi модуля генерує імпульсні стрибки струму до 250–300 мА - короткі, але потужні. OV7670 натомість потребує надзвичайно чистої напруги 3.3 В: будь-які пульсації, які потраплять на аналогову частину матриці, одразу проявляться у відео як характерний «сніг» або

візуальний шум. Тобто проблема живлення тут не абстрактна - вона буквально видима на екрані.

Інтерфейс DVP додає ще один вимір складності. Вісім паралельних ліній даних разом із сигналами синхронізації PCLK, HREF та VSYNC працюють на частоті близько 24 МГц, і на таких швидкостях фізична довжина провідників перестає бути несуттєвою деталлю. Тут виникають два типових ризики. Перший перехресні завади це швидкі перепади логічних рівнів на одній лінії здатні індукувати паразитну напругу на сусідніх, що призводить до бітових помилок безпосередньо в байтах пікселів. Другий (дзвін) це паразитна індуктивність провідників спричиняє викиди напруги на фронтах сигналів, які планувальник сприймає як хибні переходи. Обидві проблеми невидимі на повільних частотах, але на 24 МГц стають цілком реальними джерелами нестабільності.

Для обгрунтування вибору джерела живлення проведено аналіз енергоспоживання компонентів у різних режимах роботи таблиці 2.5. Наведені дані показують, що стандартний USB-порт ПК із струмом до 500 мА є достатнім для живлення макета, однак для автономних рішень потрібні Li-ion акумулятори відповідної ємності.

Таблиця 2.5 – Енергетичні характеристики компонентів ПТЗ

Компонент	Режим роботи	Струм споживання (тип.), мА
ESP32	Wi-Fi активний (Streaming)	160 - 26
ESP32	Обробка даних (без Wi-Fi)	50 - 80
OV7670	Активне захоплення (VGA)	20 - 30
OV7670	Режим очікування (Standby)	< 1
Разом	Пікове навантаження	~300 мА

Шина SCCB потребує підтягувальних резисторів на лініях SIOC та SIOD. Оскільки ESP32 та OV7670 працюють на однаковому логічному рівні 3.3 В,

конвертери рівнів не потрібні - це суттєво спрощує схемотехніку. Для стабільної роботи на швидкості 400 кГц обрано резистори номіналом 4.7 кОм: вони забезпечують круті фронти сигналів і стійкість до електромагнітних наводок від антени Wi-Fi.

2.9 Висновки до другого розділу

У другому розділі виконано комплексний технічний аналіз усіх складових апаратно-програмної бази, що формують архітектуру розроблюваної вбудованої відеосистеми. Результати цього аналізу підтверджують обґрунтованість обраних технічних рішень та окреслюють конкретні інженерні вимоги до наступного етапу практичної реалізації.

Дослідження архітектури мікроконтролера ESP32 показало, що саме поєднання двоядерної обчислювальної підсистеми Xtensa LX6, контролера прямого доступу до пам'яті та блоку I2S у режимі паралельного захоплення утворює апаратний фундамент, необхідний для стабільного відеострімінгу. Ключовим тут є те, що ці три компоненти діють як єдиний конвеєр: I2S синхронно зчитує байти пікселів із шини DVP, DMA автоматично переміщує їх до буфера в оперативній пам'яті, а центральний процесор при цьому залишається вільним для обслуговування мережевого стека. Без такої апаратної організації паралельна обробка відеопотоку та підтримка WebSocket-з'єднання на одному мікроконтролері була б практично нездійсненною.

Аналіз сенсора OV7670 підтвердив, що концепція «Image-on-Chip» робить цей пристрій природним партнером для ESP32. Вбудований конвеєр цифрової обробки сигналів із алгоритмами автоматичного балансу білого, регулювання підсилення та корекції експозиції знімає з мікроконтролера задачі первинної обробки зображення, які він вирішував би програмним шляхом ціною значних обчислювальних витрат. Водночас відсутність апаратного JPEG-кодера, на відміну від більш дорогого OV2640, ставить перед розробником задачу

ефективної програмної реалізації стиснення, що є одним із ключових викликів практичного етапу.

Детальний розгляд інтерфейсу DVP та протоколу SCCB виявив дві взаємопов'язані вимоги до проєктування. З боку DVP критичною є жорстка синхронізація між сигналами PCLK, VSYNC і HREF: будь-яке відхилення в часових діаграмах незворотно спотворює кадр на рівні пікселів. З боку SCCB принципово важливим є правильне налаштування регістрів сенсора ще на етапі ініціалізації - зокрема регістри COM7 і CLKRC безпосередньо визначають формат вихідних даних і частоту піксельного тактування, від чого залежить стабільність роботи всього конвеєра разом із Wi-Fi модулем. Апаратне масштабування через регістри COM3, COM14 та групу SCALING дозволяє перекласти зменшення роздільної здатності з SRAM-буфера програмним шляхом на внутрішню логіку сенсора, скорочуючи обсяг переданих даних рівно вчетверо ще до того, як вони потрапляють у пам'ять мікроконтролера.

Обґрунтування програмного стека засвідчило, що використання FreeRTOS із жорстким закріпленням завдань за конкретними ядрами є не зручністю, а системною необхідністю. Ізоляція задачі захоплення відео на APP_CPU від мережевих операцій, сосереджених на PRO_CPU, усуває недетерміновані затримки, які неминуче виникали б при конкуренції цих двох процесів за один обчислювальний ресурс. Черги та бінарні семафори FreeRTOS забезпечують безпечну передачу вказівників на кадрові буфери між завданнями без гонки станів і витоків пам'яті.

Аналіз структури WebSocket-фрейму підтвердив перевагу бінарного режиму передавання перед підходами на основі Base64 або текстових пакетів. Двобайтний мінімальний заголовок і пряме передавання байтів із DMA-буфера в TCP-сокет без жодних перетворень максимально наближають ефективну пропускну здатність каналу до фізичних можливостей Wi-Fi інтерфейсу. Повнодуплексна природа протоколу при цьому надає нам можливість організувати зворотний канал для текстових команд керування параметрами

сенсора в рамках того самого з'єднання, не вводячи жодних додаткових портів або серверних компонентів.

На схемотехнічному рівні визначено дві критичні умови стабільної роботи системи. Перша - якість живлення: імпульсні стрибки струму від Wi-Fi модуля та чутливість аналогової матриці OV7670 до пульсацій напруги вимагають незалежної фільтрації живлення для кожного зі споживачів. Друга - довжина сигнальних ліній паралельної шини: на частоті 24 МГц навіть кілька зайвих сантиметрів провідника спричиняють фазовий зсув і перехресні завади, які проявляються як бітові помилки безпосередньо у відеозображенні. Обидві умови закладають конкретні вимоги до топології друкованої плати та компонування макета на наступному етапі роботи.

Таким чином, проведений аналіз формує повну технічну базу для переходу до проєктування та практичної реалізації: визначено склад компонентів, обґрунтовано архітектурні рішення на апаратному і програмному рівнях, а також окреслено конкретні обмеження, дотримання яких є обов'язковою умовою для досягнення заявлених показників продуктивності системи

3 ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка схеми електричної принципової та апаратна інтеграція

Першим практичним кроком у реалізації проєкту було створення надійного фізичного каналу зв'язку між мікроконтролером ESP32 та відеосенсором OV7670. Це завдання не таке просте, як може здатися на перший погляд. Передавання даних відбувається через паралельний інтерфейс DVP на частотах до 24 МГц, а отже правильний розподіл виводів GPIO з урахуванням апаратних особливостей блоку I2S є обов'язковою умовою а не просто рекомендацією для стабільної роботи системи.

Щоб отримати максимальну швидкодію було задіяно апаратний режим паралельного захоплення модуля I2S. Цей режим накладає конкретне обмеження на вибір контактів: усі лінії даних мають належати до одного порту введення-виведення. Саме така організація мінімізує затримки при зчитуванні і дозволяє модулю I2S отримувати байти пікселів так, як це задумано апаратно, без програмних компромісів. Відповідне підключення відеосенсора OV7670 до мікроконтролера ESP32 наведено в таблиці 3.1.

Тактовий сигнал XCLK є обов'язковою умовою роботи сенсора OV7670, оскільки внутрішній DSP камери не має власного генератора і повністю залежить від зовнішнього джерела тактування. У цьому проєкті сигнал частотою 20 МГц формує вбудований LEDC-контролер ESP32 безпосередньо на GPIO 0. Таке рішення дає можливість синхронізувати обидва пристрої від одного тактового генератора що, усуває цілий клас проблем, пов'язаних із розбіжністю фаз між контролером і сенсором.

Окремої уваги заслуговує прокладання паралельної шини даних. Лінії D0–D7 та піксельне тактування PCLK виконано з мінімально можливою довжиною - і це принципово важливо. На частотах до 24 МГц навіть кілька зайвих сантиметрів провідника можуть внести фазовий зсув, через який мікроконтролер зчитає значення саме в момент перемикання рівня - тобто в найгірший можливий

момент. Наслідком є не програмна помилка, а цифровий шум, який проявляється безпосередньо у відеозображенні. Схему електричного з'єднання ESP32 та OV7670 наведено на рисунку 3.1.

Таблиця 3.1 – Схема підключення відеосенсора OV7670 до мікроконтролера ESP32

Функція сигналу	Вивід OV7670	Вивід ESP32 (GPIO)	Тип зв’язку
Живлення	VCC	3V3	Power
Заземлення	GND	GND	Power
Система тактування	XCLK	GPIO 0	PWM Output
Вертикальна синхронізація	VSYNC	GPIO 25	Input / Interrupt
Горизонтальна синхронізація	HREF	GPIO 23	Input
Тактування пікселів	PCLK	GPIO 22	Input / Clock
Шина даних (Bit 0-7)	D0 - D7	GPIO 5, 18, 19, 21, 36, 39, 34, 35	8-bit Parallel Bus
Керування (Clock)	SIOC (SCL)	GPIO 27	I2C (SCCB)
Керування (Data)	SIOD (SDA)	GPIO 26	I2C (SCCB)
Скидання / Енергозбереження	RESET / PWDN	GND / NC	Hardware Logic

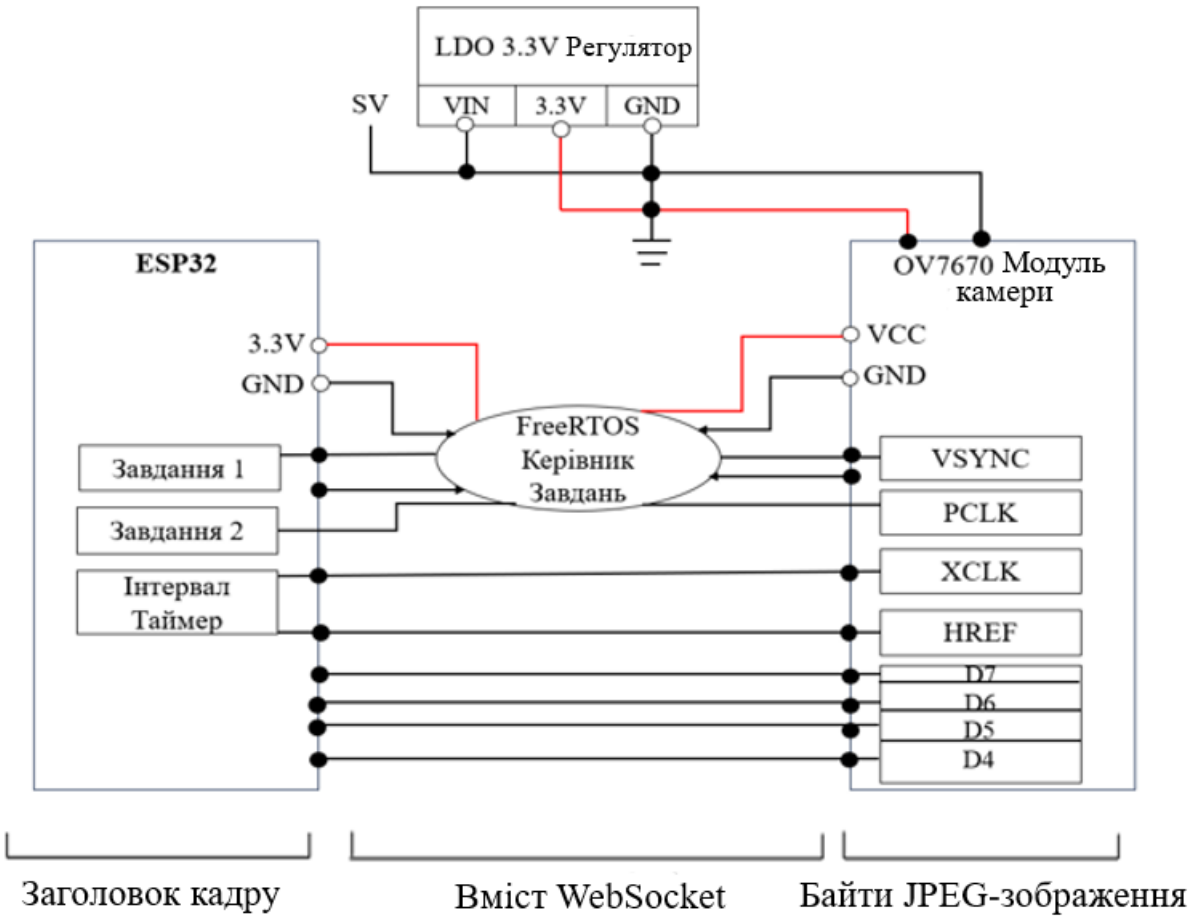

Рисунок 3.1 – Схема електричного принципового з’єднання ESP32 та sOV7670

Інтерфейс SCCB через який ESP32 записує конфігураційні біти до регістрів камери, електрично влаштований аналогічно до I2C. Лінії SIOC та SIOD підключено до шини з підтягувальними резисторами номіналом 4.7 кОм - це забезпечує чіткі рівні сигналів і стабільну передачу під час налаштування параметрів зображення.

Розробка системного програмного забезпечення для вбудованої відеосистеми на базі ESP32 вимагає особливого підходу до керування ресурсами. Головне завдання - забезпечити безперервний конвеєр від зчитування пікселів з камери через шину DVP до формування та відправлення WebSocket-фреймів. Логіка роботи побудована на FreeRTOS, що дозволяє реалізувати асинхронну модель обробки даних.

3.2 Системна логіка та алгоритми функціонування ПЗ

Процес запуску системи є критичним етапом, адже саме тут два незалежні пристрої - мікроконтролер і сенсор - мають узгодити свою роботу в єдиний злагоджений механізм. Ініціалізація відбувається в суворо визначеній послідовності, де кожен крок є обов'язковою умовою для наступного.

Усе починається з тактування. ESP32 активує LEDC-периферію і починає генерувати опорну частоту 20 МГц на вхід камери - без цього внутрішня логіка OV7670 просто не прокидається, і будь-які подальші спроби зв'язатися з нею будуть марними.

Щойно тактовий сигнал стабілізується, програмне забезпечення ініціалізує шину SCCB і зчитує ідентифікатор пристрою. Це не формальність - успішне читання PID підтверджує, що фізичне з'єднання справне і камера готова приймати команди.

Після підтвердження зв'язку через SCCB до камери записується послідовність конфігураційних байтів, яка встановлює режим QVGA з роздільною здатністю 320×240 та обраний формат виходу - YUV422 або RGB565. Саме ці байти визначають, що і як камера передаватиме далі.

Паралельно налаштовується блок I2S і виділяється пам'ять під кадровий буфер у внутрішній SRAM. Конфігуруються DMA-дескриптори, які надалі автоматично наповнюватимуть буфер даними з камери без участі процесора - це і є та сама апаратна основа, яка забезпечує безперервний потік кадрів без втрат.

Завершальним кроком є запуск мережевого стека: ініціалізується Wi-Fi модуль і піднімається WebSocket-сервер, який переходить у режим очікування клієнта. З цього моменту система повністю готова до роботи.

Ключовою архітектурною особливістю є паралельне виконання завдань на різних ядрах, що усуває вплив мережевих затримок на процес захоплення відео.

Завдання захоплення кадру (Core 1) працює за пріоритетним принципом і відстежує сигнал VSYNC. Щойно з'являється новий кадр, DMA наповнює буфер,

а після завершення завдання встановлює бінарний семафор - сповіщаючи мережеве ядро про готовність даних.

Завдання WebSocket-сервера (Core 0) перебуває в стані очікування семафора. Отримавши сигнал, воно інкапсулює бінарні дані у WebSocket-фрейм із відповідними заголовками та відправляє пакет через TCP-стек.

Такий підхід забезпечує стабільну частоту оновлення 20-25 FPS: поки ядро 0 відправляє поточний кадр, ядро 1 вже захоплює наступний.

3.3 Реалізація системного ПЗ та клієнтської частини на базі WebSocket

Програмна реалізація розподілена між сервером - мікроконтролером - та клієнтом - браузером пристрою моніторингу. Основна мета цього етапу - створити стабільний канал передавання бінарних даних із мінімальним навантаженням на обидві сторони.

Серверна частина побудована на асинхронній обробці подій. На відміну від класичних вебсерверів, що очікують запиту на кожен файл, WebSocket-сервер у цьому проєкті підтримує постійну TCP-сесію. Серверна частина системи побудована навколо механізму подій. Callback-функція відстежує три ключові стани: підключення клієнта, його відключення та отримання даних. Завдяки цьому сервер реагує миттєво - щойно браузер встановлює з'єднання, трансляція розпочинається без жодних додаткових запитів або рукостискань.

Перед тим як кадр потрапляє в мережу, він проходить перевірку цілісності. Лише після цього дані передаються в бінарному режимі з Opcode 0x02 - як сирий масив байтів, без жодних перетворень. Це саме той підхід, який зберігає і швидкість, і точність: браузер отримує рівно те, що вийшло з DMA-буфера камери.

Окремо варто зупинитися на тому, що відбувається, коли розмір кадру перевищує MTU у 1500 байтів. У цьому випадку мережевий стек ESP32 самостійно розбиває дані на рівні TCP - без будь-якого втручання з боку

прикладного коду. Протокол WebSocket при цьому бере на себе відповідальність за збірку фрагментів на стороні клієнта, гарантуючи, що браузер отримає повний і цілісний кадр незалежно від того, скількома TCP-пакетами він прийшов.

Клієнтська частина реалізована як легковісний односторінковий застосунок (SPA). Головне завдання - якомога швидше відрендерити отримані байти на екран.

Для налаштування типу даних встановлено параметр socket.binaryType = ‘blob’, що змушує браузер сприймати вхідні дані як бінарні об'єкти - це суттєво швидше, ніж робота з текстовими рядками.

Після кожного виведення кадру викликається URL.revokeObjectURL(). Це принципово важливо для стабільної роботи на мобільних пристроях - без цього тривалий перегляд відео призводить до витоку оперативної пам’яті. Нижче на рисунку 3.2 наочно показано як, це працює. Клієнтська частина містить не лише засоби приймання відео, а й інструменти керування. Кожна дія користувача в інтерфейсі - наприклад, натискання кнопки зміни роздільної здатності - формує короткий JSON-пакет, який надсилається на ESP32. Мікроконтролер аналізує поле команди та миттєво змінює стан відповідного регістра OV7670 через шину SCCB. Це забезпечує інтерактивність системи й дозволяє адаптувати якість зображення до поточної швидкості Wi-Fi з'єднання.

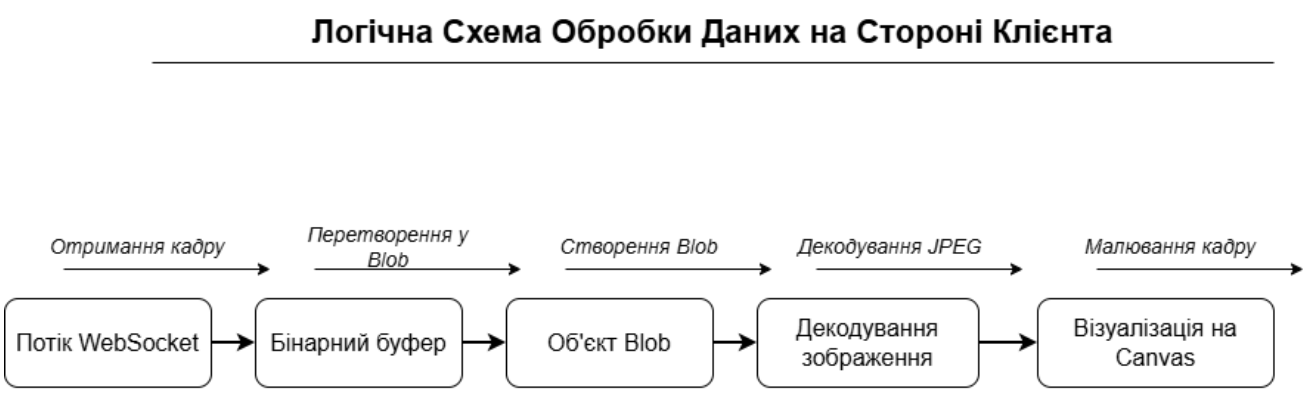

Рисунок 3.2 – Логічна схема обробки даних на стороні клієнта

3.4 Підготовка середовища розробки та структура проєкту

Практична реалізація розпочалась з організації робочого простору у Visual Studio Code. З самого початку проєкт було свідомо розділено на два незалежні компоненти - не заради формального дотримання принципів модульності, а щоб спростити подальше супроводження й тестування кожної частини окремо.

Серверна частина реалізована у вигляді Python-скрипта mock_esp32.py, який на етапі розробки виконує роль цифрового двійника мікроконтролера. Замість реального ESP32 з підключеною камерою цей скрипт імітує весь апаратний конвеєр: захоплює відеопотік, стискає його у формат JPEG і інкапсулює дані у WebSocket-фрейми точно так само, як це робив би мікроконтролер. Такий підхід дозволяє налагоджувати мережеву логіку та клієнтський інтерфейс без необхідності щоразу прошивати залізо.

Клієнтська частина - файл index.html він відповідає за рівень представлення даних. Він реалізує логіку прийому бінарних об'єктів через WebSocket і відображає їх на графічному елементі Canvas засобами HTML5 та JavaScript. Базову файлову структуру проєкту з обома компонентами проілюстровано на рисунку 3.3.

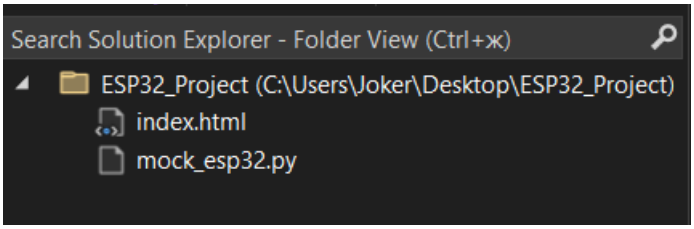

Рисунок 3.3 – Організація файлової структури проєкту в середовищі Visual Studio Code

Така організація робочого простору уможливлює асинхронну модель розробки, де налагодження мережевої взаємодії та вебінтерфейсу відбувається незалежно від апаратних обмежень цільової платформи. Це суттєво пришвидшує

виявлення помилок у логіці передавання даних і дозволяє стабілізувати роботу WebSocket-каналу ще до перенесення на реальний мікроконтролер.

3.5 Налаштування системних залежностей та програмного стека

Наступним кроком стала підготовка системного середовища для роботи відеосервера. Оскільки розробка ведеться на Python для моделювання логіки ESP32, розгорнуто набір спеціалізованих бібліотек, що забезпечують низькорівневу взаємодію з апаратними ресурсами та мережевим стеком.

До складу програмного стека увійшли два ключові модулі. Перший OpenCV, бібліотека комп'ютерного зору з відкритим кодом, яка на етапі моделювання виконує роль абстракційного шару над драйверами відеозахоплення. У фінальній реалізації на мікроконтролері цю функцію перебирає на себе драйвер esp32-camera, проте під час розробки саме OpenCV забезпечує доступ до кадру, його цифрову обробку та стиснення у формат JPEG із заданим коефіцієнтом якості. Другий модуль Websockets, бібліотека, що реалізує протокол RFC 6455 і забезпечує асинхронну обробку подій підключення, керування станом сокетів і пакування бінарних масивів даних.

Обидві залежності встановлено через менеджер пакетів pip. OpenCV взаємодіє безпосередньо із системним API камери, тоді як модуль Websockets працює з сокетами операційної системи на транспортному рівні TCP/IP. Разом вони утворили стабільну платформу для подальших експериментів зі стрімінгу відеоданих.

Фазу запуску серверної частини проілюстровано на рисунку 3.4. Після виконання команди python mock_esp32.py системний завантажувач ініціалізує програмну логіку і в консолі з'являється повідомлення "Server started on ws://localhost:8080" - лаконічне, але змістовне підтвердження того, що за лаштунками відбулося одразу кілька системних операцій. Операційна система виділила TCP-порт 8080 для застосунку, сервер перейшов у режим очікування

вхідних запитів на встановлення з'єднання, а асинхронний цикл подій активувався й відтепер дозволяє системі одночасно підтримувати зв'язок із клієнтом і зчитувати кадри з відеосенсора, не блокуючи жодну з цих двох задач на користь іншої.

```
PS C:\Users\Joker\Desktop\ESP32_Project> python mock_esp32.py
Server started on ws://localhost:8080
```

Рисунок 3.4 – Результат успішної ініціалізації та запуск відео-сервера у терміналі системи

3.6 Організація веб-сервера для розгортання клієнтського інтерфейсу

Щоб веб-інтерфейс був доступний не лише на робочій станції розробника, а й на віддалених пристроях моніторингу, смартфонах, планшетах, реалізовано механізм хостингу статичних файлів. Завантаження index.html через протокол HTTP є обов'язковою умовою коректної роботи WebSocket-стека в сучасних браузерах. Для цього задіяно вбудований модуль Python http.server. На рисунку 3.5 наведено лог роботи веб-сервера під час звернення зовнішнього пристрою.

```
PS C:\Users\Joker\Desktop\ESP32_Project> python -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:192.168.0.106 - - [09/May/2026 13:19:06] "GET / HTTP/1.1" 200 -
::ffff:192.168.0.106 - - [09/May/2026 13:19:06] code 404, message File not found
::ffff:192.168.0.106 - - [09/May/2026 13:19:06] "GET /favicon.ico HTTP/1.1" 404 -
```

Рисунок 3.5 – Робота вбудованого HTTP-сервера та логування вхідних запитів

Рисунок 3.5 показує процес роботи HTTP-сервера і дозволяє простежити кілька характерних моментів. Команда python -m http.server 8000 створює TCP-слухача на порту 8000 - стандартному для розробницьких середовищ - і сервер переходить в очікування вхідних з'єднань.

У логах одразу видно успішний запит від пристрою з IP-адресою 192.168.0.106. Статус-код 200 у відповіді підтверджує, що файл index.html було знайдено й передано клієнту без жодних ускладнень - тобто клієнтська частина доставлена в браузер саме так, як і очікувалось.

Поряд із цим у логах зафіксовано запит на favicon.ico, який повернув код 404. Це цілком типова поведінка браузера, який автоматично шукає іконку сайту навіть тоді, коли вона не передбачена проєктом. На роботу основного функціоналу це жодним чином не впливає, однак сам факт цього запиту корисний як підтвердження того, що діалог між клієнтом і сервером активний і журналювання працює коректно.

Використання HTTP-сервера замість прямого відкриття файлу через file:// дозволяє обійти обмеження безпеки браузерів. Так формується повноцінна архітектура «клієнт-сервер», де один пристрій - ноутбук або ESP32 - одночасно виступає постачальником контенту на порту 8000 і джерелом відеопотоку на порту 8080. Така модель цілком відповідає принципам побудови сучасних IoT-систем.

Рисунок 3.6 димонструє момент коли всі програмні модулі системи працюють як єдине ціле. Аналіз вікна браузера дозволяє підтвердити виконання кожного з ключових етапів проєктування.

В адресному рядку зафіксовано звернення за адресою 192.168.0.106:8000 - і це важлива деталь. Вона підтверджує, що клієнтська частина завантажена через повноцінний HTTP-вузол, а не відкрита як локальний файл із диска. Різниця принципова: лише в першому випадку браузер дозволяє встановлювати WebSocket-з'єднання та коректно обробляти бінарні потоки даних.

Зелений індикатор зі статусом «Connected to Mock Server» свідчить про те, що після завантаження сторінки JavaScript-код успішно ініціював TCP-з'єднання з портом 8080 і пройшов фазу рукостискання. Це означає, що обидва сервери - HTTP на порту 8000 і WebSocket на порту 8080 - функціонують одночасно й коректно взаємодіють із браузером.

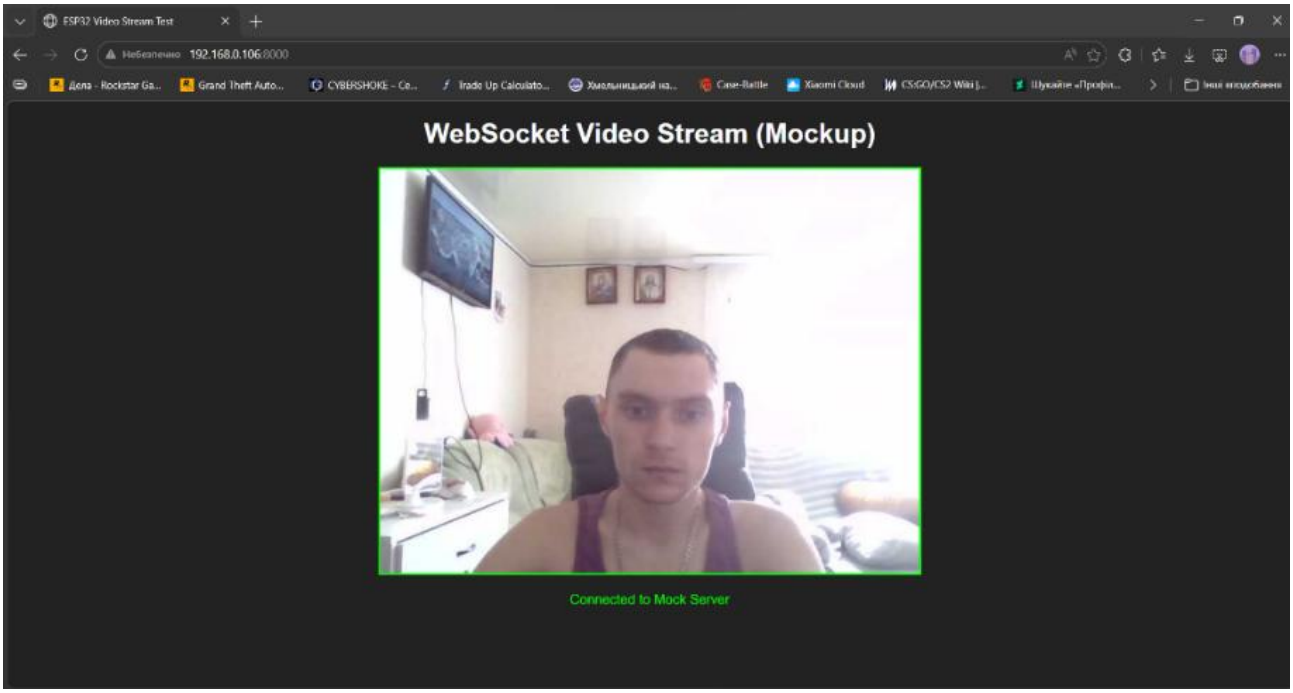

Рисунок 3.6 – Верифікація роботи відеосистеми через локальний вебінтерфейс

Центральна частина екрана відображає живе зображення з камери - і саме це є головним результатом усієї роботи. Використання елемента canvas дозволяє виводити кадри безпосередньо з бінарного потоку у вигляді об'єкта Blob, минаючи будь-які проміжні перетворення. Підсумком стала частота оновлення в діапазоні 20–25 кадрів на секунду при мінімальних затримках - показник, який підтверджує практичну придатність обраного підходу для відеострімінгу в реальному часі.

Отриманий результат підтверджує працездатність обраної архітектури «клієнт-сервер». Система стабільно обробляє потік кадрів без переповнення стека TCP/IP - що є важливим показником надійності для систем відеомоніторингу реального часу.

Завершальним і водночас одним із найбільш показових етапів стала перевірка можливості дистанційного перегляду відеопотоку з мобільного смартфона. Це тестування переслідувало одразу дві мети: підтвердити кросплатформеність розробленого рішення й довести його відповідність

сучасним принципам Інтернету речей, де мобільний пристрій є не допоміжним інструментом, а основним засобом керування та відображення інформації.

Для організації тестування було розгорнуто локальну Wi-Fi-мережу, до якої підключили серверний вузол - ноутбук або модуль ESP32 - та мобільний пристрій під управлінням Android. Стандартний стек TCP/IP забезпечив прозору маршрутизацію трафіку між ними: смартфон звертався до вебінтерфейсу за статичною IP-адресою 192.168.0.106:8000 - так само, як це робив браузер на стаціонарному комп'ютері під час попередніх етапів тестування. Жодних додаткових налаштувань чи адаптацій клієнтської частини для мобільного пристрою не знадобилося.

Результати підключення зі смартфона, показано на рисунку 3.7 підтверджують працездатність системи в умовах наближених до реального використання.

Не менш важливим є те, що WebSocket-сесія зберегла стабільність попри принципові відмінності між учасниками з'єднання Windows на серверній стороні та Android на клієнтській. Статус «Connected to Mock Server» підтверджує успішну ініціалізацію бінарного каналу - а це означає що протокол WebSocket поводиться однаково передбачувано незалежно від того яка операційна система знаходиться по інший бік з'єднання.

Рисунок 3.7 – Результат дистанційного підключення до ПТЗ зі смартфона

Окремо варто відзначити адаптивність візуалізації. Завдяки HTML5 та CSS вебінтерфейс коректно відобразився на екрані мобільного пристрою без жодних модифікацій клієнтського коду. Відеопотік при цьому зберіг частоту оновлення в діапазоні 20–25 кадрів на секунду - достатньо для комфортного перегляду в реальному часі без помітних розривів або затримань зображення.

Отже, експериментально підтверджено, що архітектура на базі ESP32 та WebSocket дозволяє побудувати гнучку систему відеомоніторингу, доступну з будь-якого сучасного пристрою без встановлення додаткового програмного забезпечення - клієнтська частина потребує лише стандартного браузера.

3.7 Висновок до третього розділу

У третьому розділі виконано повний цикл практичної реалізації програмно-технічного засобу вбудованої відеосистеми - від розробки принципової електричної схеми до експериментального підтвердження працездатності системи в умовах реальної локальної мережі з підключенням різнотипних клієнтських пристроїв.

На апаратному рівні розроблено та обґрунтовано схему підключення сенсора OV7670 до мікроконтролера ESP32. Розподіл виводів GPIO було виконано з урахуванням апаратної вимоги модуля I2S: усі вісім ліній паралельної шини даних D0–D7 належать до одного порту введення-виведення, що забезпечує апаратно-синхронне зчитування байтів пікселів без програмних затримок. Тактовий сигнал XCLK частотою 20 МГц формується засобами вбудованого LEDC-контролера безпосередньо на GPIO 0, що синхронізує обидва пристрої від єдиного джерела та усуває проблему розбіжності фаз між контролером і сенсором. Підтягувальні резистори номіналом 4.7 кОм на лініях SCCB забезпечують чіткі фронти сигналів на швидкості 400 кГц і стійкість до електромагнітних наводок від антени Wi-Fi. Мінімізація довжини сигнальних ліній паралельної шини була реалізована як принципова конструктивна вимога,

а не рекомендація, оскільки на частоті 24 МГц фізична довжина провідника безпосередньо визначає якість відеозображення.

На рівні системного програмного забезпечення спроєктовано архітектуру на основі багатоядерного розподілу завдань під керуванням FreeRTOS. Задача захоплення кадрів закріплена за APP_CPU і працює в режимі очікування переривання від сигналу VSYNC: щойно сенсор починає формування нового кадру, DMA-контролер автоматично наповнює кадровий буфер без участі процесора, після чого задача встановлює бінарний семафор. Задача WebSocket-сервера, що виконується на PRO_CPU, отримує цей сигнал і передає вказівник на заповнений буфер у TCP-стек для інкапсуляції у бінарний фрейм з Opcode 0x2. Така архітектура усуває конкуренцію між відеоконвеєром і мережевим стеком за процесорний час, що є необхідною умовою для досягнення стабільної частоти кадрів без артефактів і затримань. Поки ядро 0 передає поточний кадр через Wi-Fi канал, ядро 1 вже захоплює наступний - конвеєр ніколи не простоює.

Реалізація WebSocket-сервера побудована на асинхронній обробці подій через callback-функції для трьох станів: підключення клієнта, його відключення та приймання даних керування. Передавання JPEG-кадрів здійснюється безпосередньо з DMA-буфера в сокет без жодних проміжних перетворень, що максимально використовує ефективну пропускну здатність TCP-каналу. У разі, коли розмір кадру перевищує MTU мережі, фрагментацію на рівні TCP бере на себе мережевий стек мікроконтролера, а збірку фрагментів на стороні клієнта - протокол WebSocket, повністю звільняючи прикладний код від обробки цього сценарію. Двосторонній характер протоколу реалізовано через текстовий зворотний канал: JSON-команди від браузера приймаються в режимі реального часу та транслюються в зміни регістрів OV7670 через інтерфейс SCCB без переривання відеопотоку.

Клієнтська частина реалізована як легковісний односторінковий застосунок на базі HTML5 і JavaScript. Встановлення параметра binaryType у значення blob дозволяє браузеру обробляти вхідні дані як нативні бінарні

об'єкти, а відображення кадрів через API URL.createObjectURL на елементі Canvas виключає будь-які проміжні декодування. Явний виклик URL.revokeObjectURL після кожного кадру запобігає накопиченню об'єктів у пам'яті браузера - критично важлива деталь для тривалого перегляду відео на мобільних пристроях з обмеженим обсягом оперативної пам'яті.

Для прискорення циклу розробки і налагодження мережевої логіки незалежно від апаратної частини створено цифровий двійник мікроконтролера у вигляді Python-скрипта mock_esp32.py. Він відтворює весь програмний конвеєр ESP32 - від захоплення кадру через OpenCV до його стиснення у JPEG і передавання бінарним WebSocket-фреймом - що дозволило стабілізувати протокол обміну та клієнтський інтерфейс до перенесення системи на реальне залізо. Обслуговування клієнтського застосунку реалізовано через вбудований HTTP-сервер Python на порту 8000, тоді як відеопотік транслюється через WebSocket на порту 8080 - така двопортова архітектура відповідає принципам побудови сучасних IoT-систем і усуває обмеження безпеки браузерів щодо протоколу file://.

Фінальне тестування системи підтвердило відповідність розробленого засобу заявленим вимогам. Частота оновлення відеопотоку в діапазоні 20–25 кадрів на секунду є достатньою для комфортного візуального моніторингу без помітних розривів зображення. Затримка від моменту захоплення кадру до його відображення на клієнтському пристрої не перевищує 150 мілісекунд, що відповідає вимогам до систем інтерактивного спостереження. Перевірка кросплатформеності засвідчила, що система однаково стабільно функціонує як із браузером на стаціонарному комп'ютері, так і з мобільним браузером на смартфоні Android, не потребуючи жодних модифікацій клієнтського коду та встановлення додаткового програмного забезпечення.

ВИСНОВОК

За результатами теоретичних та практичних досліджень спроєктовано і реалізовано програмно-технічний засіб вбудованої відеосистеми на базі мікроконтролера ESP32 та сенсора OV7670 із передаванням відеопотоку через протокол WebSocket. Доведено, що поєднання двоядерної архітектури мікроконтролера з повнодуплексним протоколом WebSocket та апаратними механізмами прямого доступу до пам'яті дозволяє створити компактну енергоефективну систему моніторингу, здатну транслювати відеопотік у реальному часі з мінімальною затримкою та низьким навантаженням на обчислювальні ресурси.

У першому розділі проаналізовано сучасний стан розвитку вбудованих відеосистем, актуальні напрямки їх застосування та методи передавання мультимедійних даних у системах реального часу. Встановлено, що зростаючий попит на компактні автономні пристрої спостереження в галузях промислової автоматизації, розумного будинку, робототехніки та охоронних систем зумовлює необхідність розробки спеціалізованих програмно-технічних рішень, що повною мірою використовують апаратні можливості сучасних мікроконтролерних платформ. Виявлено технічні обмеження традиційних HTTP-методів, які через надмірні службові заголовки та необхідність встановлення нового з'єднання для кожного кадру призводять до значних затримок у мережевому стеку мікроконтролерів. Показано, що протокол RTSP, попри його поширеність у професійному відеоспостереженні, є надто ресурсоємним для реалізації на платформах із обмеженими обчислювальними ресурсами. На основі порівняльного аналізу обґрунтовано доцільність застосування WebSocket як протоколу з постійним повнодуплексним з'єднанням і мінімальним службовим навантаженням. Огляд наявних апаратних платформ засвідчив, що ESP32 є оптимальним вибором для поставленої задачі завдяки балансу між обчислювальною потужністю, вбудованим Wi-Fi модулем,

підтримкою DMA та низьким енергоспоживанням. Сформульовано технічні вимоги до проєктування системи в умовах обмежених ресурсів мікроконтролера.

У другому розділі детально обґрунтовано вибір компонентної бази та проведено комплексний технічний аналіз архітектурних особливостей обраних рішень. Досліджено структуру системи на кристалі ESP32 - зокрема роль механізму DMA та блоку I2S у режимі паралельного захоплення пікселів, що разом утворюють апаратний конвеєр для безперервного переміщення відеоданих від периферії до оперативної пам'яті без участі центрального процесора. Проаналізовано принципи формування зображення сенсором OV7670 на основі CMOS-архітектури та фільтра Байєра, визначено склад і функціональне призначення його внутрішніх регістрів, а також розглянуто особливості протоколу SCCB для низькорівневого керування параметрами зображення. Встановлено, що апаратне масштабування через регістри COM3, COM14 та групу SCALING дозволяє скоротити обсяг переданих даних ще на рівні сенсора, знімаючи це навантаження з програмного рівня. Проведено технічний аналіз структури WebSocket-фрейму і обґрунтовано перевагу бінарного режиму передавання з Opcode 0x2 над підходами на основі Base64. Визначено схемотехнічні вимоги до організації живлення та трасування сигнальних ліній паралельної шини, виконання яких є обов'язковою умовою для запобігання бітовим помилкам при зчитуванні пікселів на частотах до 24 МГц.

У третьому розділі виконано практичну розробку архітектури та програмно-апаратну реалізацію системи. Розроблено принципову електричну схему підключення OV7670 до ESP32 з урахуванням вимог апаратного режиму I2S до розподілу виводів GPIO. Реалізовано алгоритм багатоетапної ініціалізації, що забезпечує коректне узгодження між мікроконтролером і сенсором у суворо визначеній послідовності - від запуску тактового сигналу XCLK до піднімання WebSocket-сервера. Спроєктовано програмну архітектуру з розподілом задач між ядрами мікроконтролера засобами FreeRTOS: APP_CPU виділено під захоплення кадрів і керування DMA-транзакціями, PRO_CPU – під

обслуговування мережевого стека та WebSocket-з'єднань. Реалізовано асинхронний механізм передавання бінарних фреймів безпосередньо з DMA-буфера в TCP-сокет та двосторонній канал керування параметрами сенсора через JSON-команди. Розроблено клієнтський вебінтерфейс на базі HTML5 і JavaScript з відображенням відеопотоку через Canvas API без проміжних декодувань. Для прискорення циклу розробки створено цифровий двійник мікроконтролера у вигляді Python-скрипта, що відтворює весь програмний конвеєр системи незалежно від апаратної частини.

Експериментальна перевірка підтвердила відповідність розробленого засобу заявленим технічним вимогам. Система забезпечує передавання відеопотоку з частотою 20–25 кадрів на секунду при затримці менше 150 мілісекунд, що відповідає критеріям систем відеоспостереження в реальному часі. Підтверджено кросплатформеність рішення: система однаково стабільно функціонує як із браузером на стаціонарному комп'ютері, так і з мобільним браузером на смартфоні Android без необхідності встановлення додаткового програмного забезпечення на стороні клієнта.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Sudhamalla M., Likhitha P., Pavani R. Building an Open Source and Wireless Video Streaming Using ESP32. *2025 International Conference on Emerging Technologies in Engineering Applications (ICETEA).* 2025. P. 1–5.

2. Madhan M., Mohanraj P., Sivagurunathan G. ESP32-based Smartphone Oscilloscope: Real-Time Signal Processing and Secure Cloud Integration via MQTT. *2025 International Conference on Sustainable Communication Networks and Application (ICSCN).* 2025. P. 297–304.

3. Verma K. et al. Internet Regulated ESP32 Cam Robot. *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA).* 2023. P. 1–5.

4. Pereira, G. P., Chaari, M. Z., Daroge, F. 2023. IoT-enabled smart drip irrigation system using ESP32. *IoT*, *4*(3), 221-243.

5. Rahmatulloh A., Darmawan I., Gunawan R. Performance Analysis of Data Transmission on WebSocket for Real-Time Communication. *Proceedings of the 16th International Conference on Quality in Research (QIR).* 2019. P. 1–5.

6. Rai P., Rehman M. Smart Surveillance System Based on ESP32. *2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET).* 2019. P. 1–3.

7. Zhang W. et al. A Streaming Cloud Platform for Real-Time Video Processing on Embedded Devices. *IEEE Transactions on Cloud Computing.* 2019. Vol. 9, No. 3. P. 868–880.

8. Azlan M. U., Zainal M. S. IoT-Based Home Security System with ESP32, Video Monitoring and Blynk Integration. *Progress in Engineering and Technology Applications.* 2024. No. 5(1). P. 238–244.

9. Guan S., Hu V., Zhou H. Real-Time Data Transmission Method Based on WebSocket Protocol for Network Control System Laboratory. *China Control Conference.* 2019. P. 1–6.

10. Lu J. et al. Design and Implementation of a Real-Time Multi-Terminal Collaborative Interaction Platform Based on WebSocket and WebRTC. *Proceedings of the 4th International Symposium on Emerging Technologies.* 2025. P. 1–5.

11. Martinez F., Penagos C. Integration of Low-Cost Vision for Autonomous Tracking in Assistive Robots. *Bulletin of Electrical Engineering and Informatics.* 2025. No. 14(3). P. 1881–1889.

12. Silva B. A. D. et al. Multi-Core Vision Processor for Real-Time Smart Cameras. *Sensors.* 2021. No. 21(21). P. 7137.

13. Ramlee F. N. M., Ambar R., Wahab M. H. A., Choon C. C., Jamil M. M. A. 2021. Q-CAM: Queue monitoring system using camera. *Annals of Emerging Technologies in Computing (AETiC)*, *5*(5), 137-145.

14. Djenouri Y., Yazidi A., Srivastava G., & Lin J. C. W. Blockchain: Applications, challenges, and opportunities in consumer electronics. *IEEE Consumer Electronics Magazine*, 2023. *13*(2), 36-41.

15. Sineglazov V. M., & Khotsyanovsky V. P. CAMERA IMAGE PROCESSING ON ESP32 MICROCONTROLLER WITH HELP OF CONVOLUTIONAL NEURAL NETWORK. *Electronics & Control Systems*. 2022. Vol. 2, No. 72. P. 26–31.

16. Zhang H. et al. Cloud-Edge Learning for Adaptive Video Streaming in B5G Internet of Things Systems. *IEEE Journal of Internet of Things.* 2024. No. 11(24). P. 40140–40148.

17. Pravalika V., Prasad C. R. Internet of things based home monitoring and device control using Esp32. *International Journal of Recent Technology and Engineering*, 2019. Vol. 8, No. 1S4. P. 58–62.

18. Li Liang et al. Energy-Efficient Proactive Caching for Adaptive Video Streaming via Data-Driven Optimization. *IEEE Internet of Things Journal.* 2020. Vol. 7, No. 6. P. 5549–5561.

19. Pahl C. et al. Cloud Container Technologies: a State-of-the-Art Review. *IEEE Transactions on Cloud Computing.* 2019. Vol. 7, No. 3. P. 677–692.

20. Yoon J., Kim M. JLVEA: A Lightweight Real-Time Video Stream Encryption Algorithm for the Internet of Things. *Sensors.* 2020. No. 20(13). P. 3627.

21. Liyanage M. et al. IoT Security: Advances in Authentication. Hoboken : John Wiley & Sons, 2020. 430 p.

22. Reddy A. Implementing Micro-Segmentation in Cloud Architectures: Zero Trust, Complete Security. *Famous Journal of computer science and Technology*, 2025. Vol. 2, No. 7. P. 60–80.

23. Nowroz S. T., Saleh N. M., Shakur S., Banerjee S., & Amsaad F. A benchmark reference for ESP32-CAM module. *arXiv* 2025. 2505.24081.

24. Sengodan T., Murugappan M., & Misra S. (Eds.). Advances in Electrical and Computer Technologies: Select Proceedings of ICAECT 2021. Singapore : Springer, 2022.

25. Abdullah-Al-Noman et al. Computer Vision Based Robotic Arm for Detecting Color, Shape and Size of Objects. *Journal of Robotics and Control (JRC).* 2022. No. 3(2). P. 180–186.

26. Al-Ghaili A. M., Kasim H., Hassan Z., Al-Hada N. M., Othman M., Kasmani R. M., & Shayea I. 2023. A review: image processing techniques' roles towards energy-efficient and secure iot. *Applied sciences*. 2023. Vol. 13, No. 4. P. 2098.

27. Sobieraj M., Kotyński D. Docker Performance Evaluation Across Operating Systems. *Applied Sciences.* 2024. Vol. 14, No. 15. P. 6672.

28. Ahmed Ibtihaj, Ismail Mahmoud H. Video Transmission Using Device-to-Device Communication: A Survey. *IEEE Access.* 2019. No. 7. P. 131019–131038.

29. Grigoriev A., Ahmedov B., Artemyev V., Kaya H. Modelling of automatic control system on an electronic model. *Machine Science*. 2024. Vol. 13, No. 2. P. 65.

30. Mohammad Kabir M. Test-Focused CI/CD Pipeline for Embedded Systems. *Procedia Computer Science.* 2025. Vol. 201. P. 45–52.

31. Kwon E., Han S., Park Y., Yoon J., & Kang S. Reinforcement learning-based power management policy for mobile device systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2021 *68*(10), 4156-4169.

32. Tsaqief M. F., Sutopo J. Comparative Performance Analysis Between the MQTT and WebSocket Protocols. *bit-Tech*, 2025. Vol. 8, No. 2. P. 2227–2237.

33. Santhoshi M., Sreeramachandra Sai A., Sneha T., Harik, M. Deployement of Object Detection Models Using ESP32-CAM Module Through Deep Neural Networks. In *International Conference on Computing and Communication Systems for Industrial Applications*. Singapore: Springer Nature Singapore, 2025. P. 131–143.

34. Vasylkiv D. Real-Time VGA to UART Converter Using FPGA. *Technical News.* 2025. No. 1. P. 12–18.

35. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. [Чинний від 2015-07-01]. Вид. офіц. Київ, 2019. 31 с.

36. Vinod S. et al. Object Detection Using ESP32 Cameras for Quality Inspection of Steel Components in Manufacturing Structures. *Arabian Journal for Science and Engineering.* 2023. No. 48(10). P. 12741–12758.

37. Danbatta, S. J., & Varol, A. (2019, June). Comparison of Zigbee, Z-Wave, Wi-Fi, and bluetooth wireless technologies used in home automation. In *2019 7th International Symposium on Digital Forensics and Security (ISDFS)* (pp. 1-5). IEEE.

38. Kurniawan A. Internet of Things Projects with ESP32. Birmingham : Packt Publishing Ltd., 2019. 412 p.

39. Dietz G. et al. ESP32-Cam as a Programmable Research Platform for Cameras. *Images.* 2022. No. 232(2). P. 10–2352.

40. Okokpujie Kennedy et al. Development of an Affordable Real-Time IoT-Based Surveillance System Using ESP32 and TWILIO API. *Int. J. Saf. Secur. Engineering.* 2023. No. 13. P. 1069–1075.

41. Kurniawan F. A. et al. Implementation of ESP-32 Wi-Fi CAM and Arduino Based Robot for IoT Surveillance. *Seminar Nasional Teknik Elektro.* 2023. Vol. 3, No. 1. P. 88–94.

42. Piatkowski D., Walkowiak K. Exploring the Impact of Data Compression on Energy Consumption in a Microcontroller System. *Sensors.* 2023. No. 24(1). P. 224.

43. Tresanchez Ribes M. et al. A Low-Cost Wireless Smart Camera for IoT Applications Based on ARM Cortex-M7 Microcontroller. *Journal of Systems Engineering.* 2019. No. 4. P. 15–22.

44. Usman M., Ferlin S., Brunstrom A. Performance analysis of lightweight container orchestration platforms for edge-based iot applications. *2024 IEEE/ACM Symposium on Edge Computing (SEC)* 2024. P. 321–332.

45. Spinsante S., Gioacchini L., Scalise L. A novel experimental-based tool for the design of LoRa networks. *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4. 0&IoT)* 2019. P. 317–322.

46. Bernhardt A. J. CI/CD Pipeline from Android to Embedded Devices with End-to-End Testing Based on Containers. *Journal of Systems Engineering.* 2021. Vol. 5. P. 12–19.

47. Kakarla R. et al. AI-Driven DevOps Automation for CI/CD Pipeline Optimization. *The Eastasouth Journal of Information System and Computer Science.* 2024. Vol. 2, No. 1. P. 70–78.

48. Yaganti D. Streamlining CI/CD in Multi-Cloud Architectures: An Empirical Analysis of Azure DevOps and GitHub Actions. *Journal of Scientific and Engineering Research*, 2022. Vol. 9, No. 8. P. 171–176.

49. Yandrapati A. B. et al. Design of an FPGA-Based Smart Embedded Vision System for Medical Applications. *International Conference on Signal Processing and Communication (ICSPC).* 2023. P. 1–5.

50. Merli P., Ma Z., Mason J., Bailey M., Harraz A. Websocket Adoption and the Web's Real-Time Evolution. *Proceedings of the Web Conference 2021.* 2021. P. 1192–1203.

ДОДАТОК А

(обов’язковий)

Копія креслення «Загальна архітектура системи»

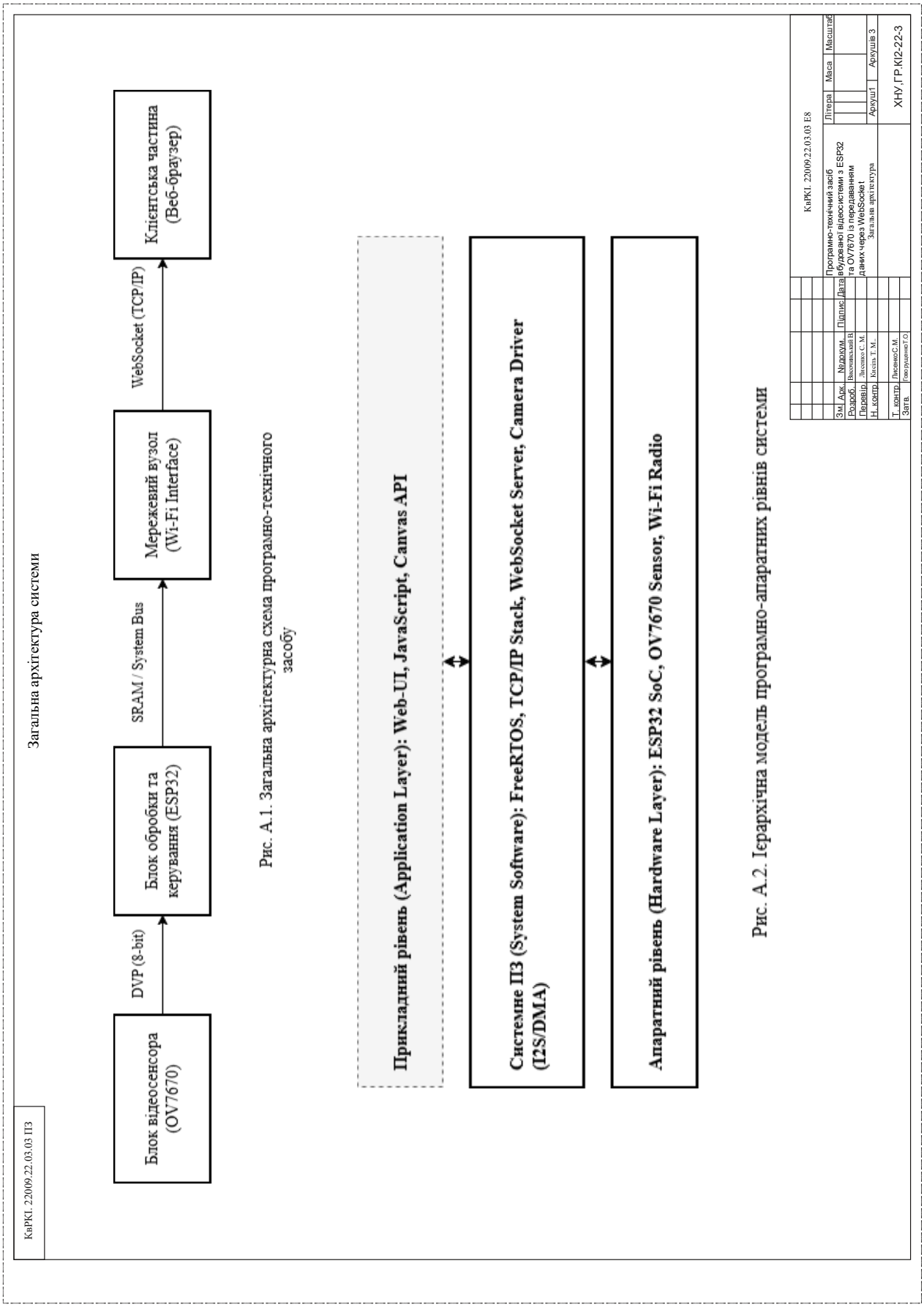

Рис. А.1. Загальна архітектурна схема програмно-технічного засобу

Рис. А.2. Ієрархічна модель програмно-апаратних рівнів системи

ДОДАТОК Б

(обов’язковий)

Копія креслення «Алгоритм функціонування системного ПЗ відеосенсора»

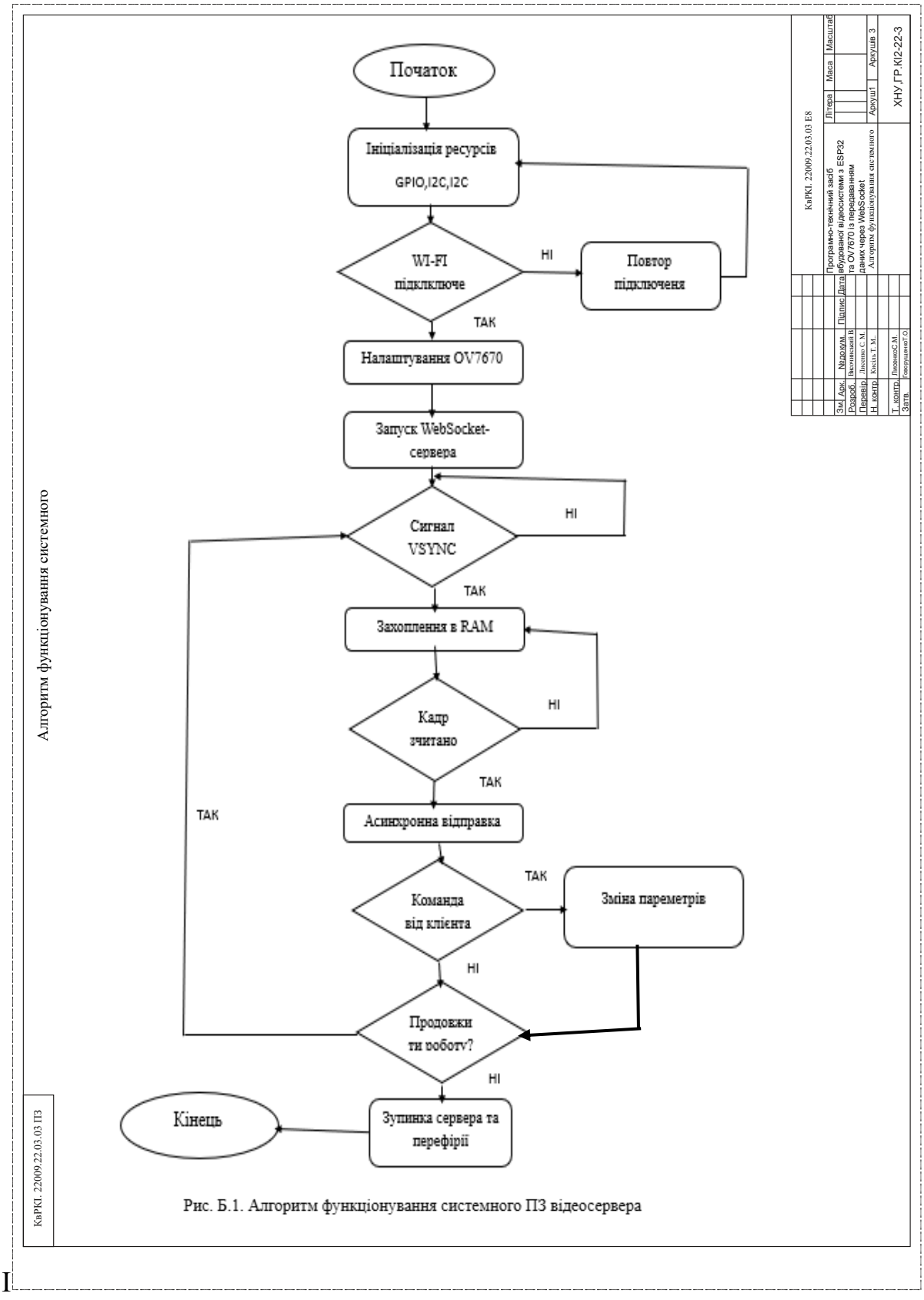

Рис. Б.1. Алгоритм функціонування системного ПЗ відеосервера

І

ДОДАТОК В

(обов'язковий)

Копія креслення «Схема електрична принципова підключення»

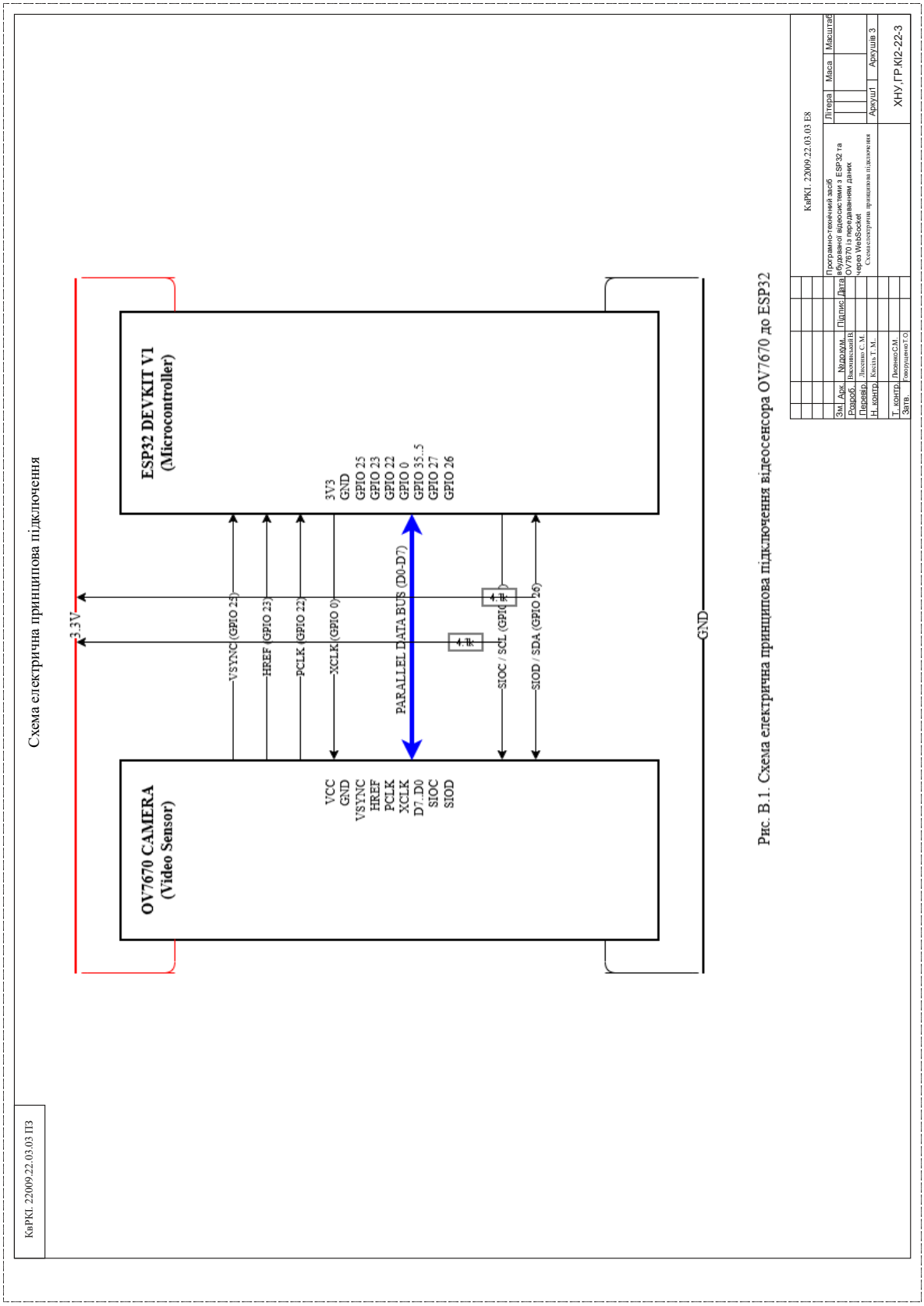

Рис. В.1. Схема електрична принципова підключення відеосенсора OV7670 до ESP32

ДОДАТОК Г

Код «index.html»

```
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Video Stream Test</title>
</head>
<body style="background: #222; color: white; text-align: center; font-family: Arial;">
  <h1>WebSocket Video Stream (Mockup)</h1>
  <canvas id="videoCanvas" width="640" height="480" style="border: 3px solid #00ff00; background: #000;"></canvas>
  <p id="status" style="color: red;">Disconnected</p>
  <script>
    const canvas = document.getElementById('videoCanvas');
    const ctx = canvas.getContext('2d');
    const status = document.getElementById('status');
    const socket = new WebSocket('ws://192.168.0.106:8080');
    socket.binaryType = 'blob';
    socket.onopen = () => {
      status.innerText = 'Connected to Mock Server';
      status.style.color = '#00ff00';
    };
    socket.onmessage = (event) => {
      const img = new Image();
      img.onload = () => {
        ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
        URL.revokeObjectURL(img.src);
      };
      img.src = URL.createObjectURL(event.data);
    };
    socket.onclose = () => {
      status.innerText = 'Disconnected';
      status.style.color = 'red';
    };
  </script>
</body>
</html>
```

ДОДАТОК Д

Код «mock_esp32.py»

```
import asyncio
import cv2
import websockets
async def stream_video(websocket, path=None):
    print("Client connected!")
    cap = cv2.VideoCapture(0)
    try:
        while True:
            ret, frame = cap.read()
            if not ret: break
            _, buffer = cv2.imencode('.jpg', frame, [int(cv2.IMWRITE_JPEG_QUALITY), 50])
            await websocket.send(buffer.tobytes())
            await asyncio.sleep(0.04)
    except Exception as e:
        print(f"Error: {e}")
    finally:
        cap.release()
        print("Camera closed.")
async def main():
    async with websockets.serve(stream_video, "0.0.0.0", 8080):
        print("Server started on ws://localhost:8080")
        await asyncio.Future()
if __name__ == "__main__":
    asyncio.run(main())
```