НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет робототехніки та приладобудування
Кафедра комп'ютерно-інтегрованих оптичних та
навігаційних систем

До захисту допущено:
Завідувач кафедри
_____Надія БУРАУ
«___»_____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерно-інтегровані системи та технології в приладобудуванні»
спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
на тему «Автоматизована система керування мультимедіа ПК»

Виконав:
студент ІІІ курсу, групи ПГ-п31
Зінін Олексій Андрійович _____
Керівник:
Доцент, к.т.н. Павловський Олексій Михайлович _____
Рецензент:
Асистент, PhD Бондарєв Денис Володимирович _____

Засвідчую, що у цієї дипломної роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2026 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут/факультет Факультет робототехніки та приладобудування
(повна назва)

Кафедра Комп'ютерно-інтегрованих оптичних та навігаційних систем
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 174 – «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
(код і назва)

Освітня програма Комп'ютерно-інтегровані системи та технології в приладобудуванні

ЗАТВЕРДЖУЮ

Завідувач кафедри КІОНС

_____ Надія БУРАУ
(підпис) (ім'я, прізвище)

«___»_____2026 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Зінін Олексій Андрійович
(прізвище, ім'я, по батькові)

1. Тема проекту «Автоматизована система керування мультимедіа ПК», керівник проекту Павловський Олексій Михайлович, к.т.н., доцент, затверджені наказом по університету від «___»_____ 2026 р. №_____

2. Строк подання студентом проекту __.06.2026 р.

3. Вихідні дані до проекту

Блок електроніки, виконавчих елементів та відповідного програмного та алгоритмічного забезпечення, що здійснюють керування мультимедіа ПК.

4. Перелік завдань, які потрібно розробити:

Вступ. Огляд апаратних контролерів для керування мультимедіа ПК. Розробка прототипу контролера. Вибір компонентів, розробка технічного та прикладного програмного забезпечення. Складання прототипу. Випробування макетного зразка.

5. Перелік графічного (ілюстративного) матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Складальне креслення (А1), Робоче креслення деталей (А3 - 2 шт), Схема алгоритму (А1), Схема електрична принципова (А1), Структурна схема (А1), 3D-модель (А1), ілюстрації, що додатково розкривають суть проекту.

6. Дата видачі завдання 24.02.2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання Дипломного проекту	Термін виконання етапів проекту	Примітка
1	Аналіз завдання до дипломного проекту	10.05.2026 – 20.05.2026	
2	Огляд існуючих рішень	21.05.2026 – 23.05.2026	
3	Вибір та обґрунтування елементів	24.05.2026 – 28.05.2026	
4	Проектування приладу	29.05.2026 – 02.06.2026	
5	Програмування приладу	03.06.2026 – 04.06.2026	
6	Аналіз працездатності	04.06.2026 – 05.06.2026	
7	Оформлення пояснювальної записки та графічних документів	06.06.2026 – 07.06.2026	

Студент _____ Олексій ЗІНІН
(підпис) (ім'я, прізвище)

Керівник проекту _____ Олексій ПАВЛОВСЬКИЙ
(підпис) (ім'я, прізвище)

Відомість дипломного проєкту

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДПБ.ПГп-31. 03.1730.01.000.ПЗ	Пояснювальна записка	113	
3	А1	ДПБ.ПГп-31. 03.1730.01.000.СК	Складальний кресленик	1	
4	А3	ДПБ.ПГп-31. 03.1730.01.001	Основа	1	
5	А3	ДПБ.ПГп-31. 03.1730.01.002	Кришка	1	
6	А1	ДПБ.ПГп-31. 03.1730.01.003	Тривимірна модель	1	
7	А1	ДПБ.ПГп-31. 03.1730.02	Електрична принципова схема	1	
8	А1	ДПБ.ПГп-31. 03.1730.03	Блок схема	1	
9	А1	ДПБ.ПГп-31. 03.1730.04	Структурна схема	1	

АНОТАЦІЯ

Дипломний проєкт присвячений розробці апаратного контролера для керування мультимедіа персонального комп'ютера. Метою проєкту є створення настільного пристрою, який бере на себе ту частину роботи, де миша й клавіатура надто повільні: одночасне регулювання гучності окремих застосунків і виклик заздалегідь заданих макросів. Подібні задачі типові для роботи із засобами автоматизованого проєктування, відеомонтажу, обробки звуку та потокового мовлення.

У ході роботи проведено огляд існуючих комерційних та аматорських рішень, сформовано перелік функціональних вимог, розроблено принципову електричну схему та виконано розведення друкованої плати. Спроєктовано корпус для тривимірного друку, написано мікропрограму мікроконтролера й супутню програму для персонального комп'ютера. Працездатність зібраного макета перевірено експериментально.

Результати дослідження продемонстрували, що повноцінний апаратно-програмний контролер мультимедіа реалізується на доступній елементній базі. Мікропрограму та супутню програму опубліковано з відкритим вихідним кодом, що дозволяє користувачеві адаптувати поведінку пристрою під власні робочі сценарії. Працездатність прототипу підтверджена серією експериментів.

ABSTRACT

The thesis project is devoted to the development of a hardware controller for personal computer multimedia control. The aim of the project is to build a desktop device that takes over the part of the workflow where a mouse and keyboard are too slow: simultaneous adjustment of individual application volume levels and execution of predefined macros. Such tasks are typical for computer-aided design tools, video editing, audio production and live streaming.

In the course of the work, a review of existing commercial and amateur solutions was conducted, functional requirements were formulated, and an electrical schematic was developed together with the layout of a printed circuit board. A housing was designed for 3D printing, microcontroller firmware and a companion program for the personal computer were written. The performance of the assembled prototype was verified experimentally.

The results demonstrated that a full-featured hardware-software multimedia controller can be implemented on an accessible component base. The firmware and companion program are released as open source, allowing users to adapt the device's behavior to their own workflows. The performance of the prototype was confirmed through a series of experiments.

ЗМІСТ

ВСТУП

Робочий персональний комп'ютер сьогодні рідко зайнятий лише однією задачею. Користувачі засобів автоматизованого проєктування, відеомонтажу, обробки звуку та потокового мовлення щодня працюють із кількома застосунками одночасно, кожен зі своєю гучністю, гарячими клавішами та налаштуваннями. Стандартні засоби операційної системи — мікшер гучності та клавіатурні скорочення — для такого сценарію не дуже зручні: щоб змінити рівень одного застосунку, доводиться шукати потрібний повзунок мишею, а складні комбінації клавіш не завжди легко запам'ятати. Це створює потребу в окремих апаратних органах керування.

Подібні пристрої давно існують на ринку: Elgato Stream Deck із програмованими LCD-кнопками, аудіомікшер TC-Helicon GoXLR із фейдерами для звукових шин та відкритий проєкт deej із лінійними потенціометрами на основі плати Arduino. Жоден з них при цьому не покриває одночасно потребу у великій кількості поворотних регуляторів, наборі програмованих кнопок, відкритому програмному забезпеченні та доступній ціні. Цей розрив і визначає актуальність розробки власного рішення.

Метою дипломного проєкту є розробка апаратного контролера для керування мультимедіа персонального комп'ютера, який об'єднує в одному настільному пристрої органи плавного керування гучністю та програмовані кнопки для виклику макросів. Завдяки апаратній підтримці інтерфейсу USB HID у обраному мікроконтролері пристрій може працювати як зі спеціалізованим драйвером, так і самостійно — у режимах HID-клавіатури та ігрового контролера — без жодного встановленого на комп'ютері програмного забезпечення.

РОЗДІЛ 1
ОГЛЯДОВА ЧАСТИНА

1.1 Передумови появи пристроїв керування мультимедіа

Робочий персональний комп'ютер рідко виконує одну задачу. У звичайний робочий день одночасно відкриті Zoom із колегами, Spotify із фоновою музикою, OBS пише екран, Discord ловить повідомлення з робочих серверів, а поруч може крутитися програма для запису музики або таймлайн відеоредактора. Кожна з цих програм має власний рівень гучності, і керувати ними системними засобами незручно. Стандартний мікшер гучності Windows (рис. 1.1) дозволяє рухати повзунки лише мишею, у вкладеному вікні налаштувань, яке ще треба відкрити. Під час прямого ефіру чи робочого дзвінка це означає чотири-п'ять кліків там, де мав би бути один рух рукою [1].

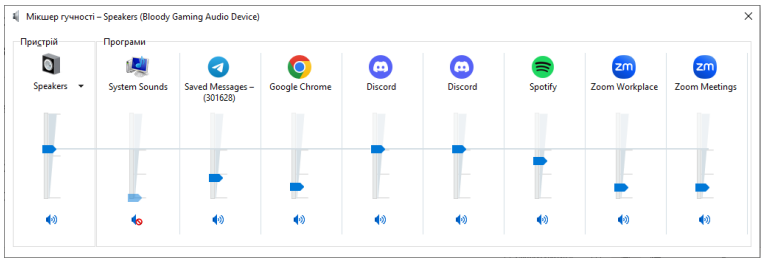


Рисунок 1.1 - Стандартний мікшер гучності Windows 10

Окрема історія - професійне програмне забезпечення. У програмах систем автоматизованого проєктування (САПР), у відеоредакторах, у програмах для роботи зі звуком меню розгалужені на десятки рівнів, а кількість гарячих клавіш доходить до сотень. Миша годиться для точкового вибору, але плавно крутити параметр - наприклад, фокусну відстань камери в Blender чи рівень окремого аудіоканалу в Reaper - нею некомфортно. Поворотний енкодер дає тактильний відгук і дозволяє керувати значенням не

дивлячись на екран, що ближче до роботи з фізичним пультом, ніж до роботи з курсором [4].

Конкретніший приклад - обробка фотографій у Adobe Lightroom. Базовий цикл редагування фотографа містить десятки повторюваних коригувань: експозиція, контраст, чорна точка, біла точка, температура, відтінок, насиченість окремих кольорів. Кожен з цих параметрів - це числове значення у вузькому діапазоні, яке зручніше обертати, ніж тягнути мишею по екрану. Аналогічна історія у Adobe Premiere і DaVinci Resolve, де колорист цілий день рухає окремі параметри коригування зображення - підняти середні тони, опустити тіні, додати теплоти у пейзаж. Без апаратного контролера усе це робиться невеликим перетягуванням віртуального повзунка мишею, що швидко стомлює руку і збільшує ризик «промазати» на пів значення.

Сюди ж лягає сценарій верстатів із числовим програмним керуванням (ЧПК). У таких верстатах оператор іноді мусить вручну переміщати інструмент по осях X, Y, Z, поворотній осі A та поворотній осі B - це називається ручним наведенням. Промислові виносні пульти ручного керування з кількома осями коштують непристойних грошей, а вісім енкодерів цілком покривають типовий 4-5-осьовий верстат із запасом: окремий регулятор на лінійне переміщення кожної осі плюс окремі ручки на швидкість подачі та оберти шпинделя. Зрозуміло, що потреба в апаратному контролері з'явилась не сьогодні і комерційні виробники її закривають уже років двадцять. Питання у тому, наскільки вдало.

Окрім професійного програмного забезпечення, є цілий пласт повсякденних сценаріїв, де фізичні елементи керування виявляються зручнішими за мишу і клавіатуру. Перший це стрімінг і прямі трансляції. Стрімеру в OBS Studio за час однієї трансляції доводиться десятки разів перемикати сцени (камера ведучого, ігровий процес, перерва, гість у дзвінку), вмикати й вимикати мікрофон, регулювати гучність голосу гостя відносно музики й гри. Усе це треба робити швидко, не виходячи з гри і не привертаючи

увагу глядачів до самого процесу керування трансляцією. Другий, відеодзвінки. У Zoom, Microsoft Teams і Google Meet окрема кнопка «вимкнути мікрофон» рятує не одну зустріч, але вмонтована в інтерфейс програми вона часто прихована. Третій сценарій, звичайна робота з браузером, де іноді корисно мати окрему кнопку для «вимкнути звук у вкладці», не шукаючи мишею кнопку на самій вкладці.

Першими у цей сегмент зайшли стрімери і відеоредактори, для яких різниця між «два кліки мишею» і «одна кнопка» прямо впливає на якість трансляції. Найвідоміший комерційний продукт - Elgato Stream Deck - вийшов на ринок ще у 2017 році. У 2019 році з'явився GoXLR - апаратний мікшер від TC-Helicon, орієнтований на роботу зі звуком. Паралельно з ними розвивався ентузіастський рух: розробник Omri Harel опублікував у 2018 році відкритий проєкт deej, який дозволяє за десяток доларів зібрати фізичний мікшер на основі Arduino [3]. Усі три це різні ніші й різні цінові категорії, але жоден не закриває макроси й керування гучністю одночасно, та ще й у бюджетному форматі. Саме цей розрив і породжує постійний потік саморобних проєктів у спільнотах ентузіастів, бо кожен намагається зібрати «свій» ідеальний контролер під свої сценарії використання.

1.2 Основні поняття та визначення

1.2.1 Мікшерна панель та системний мікшер

Мікшерна панель (мікшерний пульт, мікшер) - пристрій для зведення кількох аудіосигналів в один або кілька вихідних каналів. У звукорежисурі це громіздкий пульт з десятками каналів, де кожен має фейдер гучності, регулятори еквалайзера, панораму й допоміжні шини [2]. Загальний вигляд аналогового мікшера студійного класу наведено на рисунку 1.2.

У контексті персонального комп'ютера термін «мікшер» вживають у двох значеннях. Перше - програмний мікшер операційної системи (Windows Sound Mixer на рис. 1.1, Audio MIDI Setup на macOS, PulseAudio або PipeWire

на Linux), який дозволяє регулювати рівень кожного запущеного додатка окремо, але виключно через графічний інтерфейс. Друге - апаратний мікшер для ПК, який створює в системі віртуальні аудіопристрої й виносить керування ними на фізичні фейдери або енкодери. До другої категорії й належать пристрої, що розглядаються далі в цьому розділі.

Рисунок 1.2 - Аналоговий мікшерний пульт студійного класу

Між цими двома мікшерами є важлива різниця в способі взаємодії з користувачем. Програмний мікшер вимагає від людини щонайменше двох переключень: спочатку згорнути активний додаток або перейти на іншу панель, потім знайти потрібний повзунок серед запущених програм і лише після цього змінити рівень. Апаратний мікшер усуває обидва ці кроки: рука вже знає, де яка ручка, поглядом контролювати не треба, активний додаток на екрані не змінюється. У сценаріях прямого ефіру, відеодзвінка чи запису музики ця різниця стає принциповою, бо чим менше відволікань під час робочого процесу - тим менше помилок і вищий темп.

1.2.2 Поворотний енкодер

Поворотний (інкрементальний) енкодер - електромеханічний датчик, що перетворює обертання вала в дві імпульсні послідовності з фазовим зсувом 90° (квадратурний сигнал). За напрямком зсуву можна визначити, у який бік

крутиться вал, а за кількістю імпульсів - на скільки кроків. На відміну від потенціометра, енкодер не має крайніх положень і обертається безкінечно [4].

У комерційних контролерах енкодер часто суміщають з тактовою кнопкою всередині вала: натискання дозволяє реалізувати додаткову функцію, наприклад швидке вимкнення звуку поточного каналу або переключення параметра, яким керує цей самий регулятор. Електрично енкодери підключають як набір цифрових ліній (зазвичай дві фази та кнопка), тому їх легко зчитувати мікроконтролером. Для людини обертовий регулятор зручніший за повзунок там, де треба плавно змінювати значення без обмеження діапазону: наприклад, гучність, яскравість, прокручування таймлайну чи переміщення по списку треків.

Окремо варто згадати тактильний відгук. У дешевих енкодерах вал має невеликі фіксовані позиції, які відчуваються пальцем як легкі клацання - це і є той самий «крок». У дорожчих моделях фіксованих позицій немає або кількість позицій можна налаштовувати, що дає відчуття плавного, безперервного обертання. Для регулювання гучності кращим є варіант із фіксованими позиціями: користувач за кількістю клацань може приблизно оцінити, наскільки він зрушив значення, навіть не дивлячись на екран. Для прокручування довгого списку чи плавного руху по таймлайну зручнішим є варіант без фіксації, бо інакше відчуваються рясні мікропоштовхи.

1.2.3 Макроси

Макрос - це коли одна кнопка робить більше, ніж один клік. Натиснув - пристрій по черзі надсилає Ctrl+Alt+M, чекає 200 мс, потім перемикає сцену в OBS, а поруч ще зупиняє запис у програмі для роботи зі звуком. Для апаратного контролера макрос виглядає як послідовність дій, жорстко прив'язана до фізичної кнопки на корпусі: натискаєш - отримуєш результат, без меню, без миші, без пошуку поєднання клавіш у пам'яті [5].

На практиці макроси закривають саме ті місця, де миша й клавіатура надто повільні. Класика: миттєве вимкнення мікрофона під час дзвінка,

перемикання сцени у стрімі, вставка тексту без форматування, виклик складних поєднань клавіш Premiere, Lightroom чи After Effects на кшталт «застосувати пресет і перейти до наступного кадру». Для людини, яка проводить за такими програмами повний робочий день, виграш у часі помітний - і саме тому ринок апаратних контролерів узагалі існує.

Складніший різновид макросу - умовний, коли одна кнопка робить різні речі залежно від поточного стану системи. Наприклад, одна й та сама кнопка вмикає або вимикає мікрофон залежно від того, який його поточний стан, або вибирає сцену в OBS залежно від того, який додаток активний у фоновому режимі. Реалізувати такі сценарії суто на боці пристрою важко - пристрій не знає внутрішнього стану додатків, - тому умовну логіку зазвичай переносять у супутню програму на стороні комп'ютера. Саме такий розподіл відповідальності використовується у моєму проєкті: пристрій повідомляє про факт натискання, а програма на ПК уже вирішує, яку дію виконати з огляду на контекст.

1.2.4 Системний звуковий стек персонального комп'ютера

Окремий важливий шар, без розуміння якого складно зрозуміти роботу будь-якого апаратного мікшера для ПК, - це системний звуковий стек операційної системи. У Windows починаючи з версії Vista (2007) використовується системний звуковий рушій під назвою Windows Audio Session API (скорочено WASAPI). Він влаштований так: кожна програма, що відтворює звук, отримує власну окрему звукову сесію. Громадячи всі сесії в одну вихідну доріжку, операційна система ділить кожний канал на окрему фізичну реалізацію - за рахунок цього стає можливим керувати гучністю кожної програми окремо, не торкаючись інших. Саме цей шар і використовує стандартний мікшер Windows (рис. 1.1), а також будь-яка стороння програма, яка хоче змінювати рівень якогось додатка.

У macOS аналогічну задачу виконує підсистема Core Audio, у Linux це звукові сервери PulseAudio або новіший PipeWire. Для всіх трьох операційних

систем існують відкриті бібліотеки, які приховують деталі системних викликів за зрозумілим програмним інтерфейсом. У Windows це бібліотека pycaw для мови Python, у Linux libpulse або pipewire-rs, у macOS стандартні системні фреймворки. Для розробника апаратного контролера це означає одне: щоб керувати гучністю окремого додатка, не обов'язково писати власний драйвер. Достатньо запустити невелику програму-посередник, яка отримує події з пристрою через звичайний послідовний порт або інтерфейс USB HID, а потім викликає відповідну функцію системного звукового рушія.

Альтернативний підхід полягає в тому, щоб створювати власні віртуальні аудіопристрої. Саме так зроблено в GoXLR: його драйвер реєструє у Windows чотири окремих віртуальних звукових пристрої, і користувач у налаштуваннях Windows прив'язує кожен додаток до потрібного віртуального пристрою. Перевага підходу: повний контроль над звуковими шинами і можливість збирати кілька доріжок «для себе» і «для аудиторії» окремо. Недолік: фіксована кількість каналів і потреба у власному закритому драйвері. У моєму проєкті обрано перший підхід (керування існуючими сесіями WASAPI), бо він гнучкіший і не вимагає системних прав адміністратора.

1.3 Аналіз існуючих рішень

1.3.1 Elgato Stream Deck

Рисунок 1.3 - Сімейство пристроїв Elgato Stream Deck

Stream Deck - серія програмованих макропадів від компанії Elgato (з 2018 року - підрозділ Corsair). Перша модель з'явилась у 2017 році й відтоді стала фактичним стандартом серед стрімерів та авторів відеоконтенту [6]. Зовнішній вигляд лінійки наведено на рисунку 1.3.

Головна особливість Stream Deck полягає в тому, що під кожною кнопкою сховано власний дисплей. Розбирання, опубліковане iFixit, показує, що насправді всередині розміщується один великий рідкокристалічний екран, а зверху на нього накладено матрицю прозорих квадратних клавіш із пружинним механізмом і легким клацанням [7]. Драйвер ділить площу екрана на сегменти й виводить у кожен сегмент окрему іконку, текст чи індикатор. Корпус пластиковий, з нахиленою підставкою, підключення через USB-C (у старіших моделях micro-USB) у режимі складеного пристрою USB HID. Це дозволяє пристрою з'являтись в системі і як клавіатура (для надсилання макросів), і як спеціалізований пристрій виробника (для оновлення картинок на кнопках).

Користувач у настільній програмі Stream Deck «розкладає» дії по сітці кнопок: одна клітинка - одна дія. Дією може бути запуск програми, перемикання сцени OBS, надсилання макроса з кількох клавіш, відкриття веб-адреси, відправлення мережевого запиту тощо. При натисканні на кнопку пристрій повідомляє програму на ПК, яка виконує прив'язану дію. Картинку для конкретної кнопки програма формує сама і вантажить на пристрій через USB. Завдяки цьому Stream Deck має режим «папок»: одна кнопка може відкривати нову сторінку з 14 (або більше) іншими, а ті - наступні підпапки. Теоретично кількість дій необмежена, на практиці глибокі ієрархії доводиться запам'ятовувати окремо.

Сильна сторона Stream Deck - не залізо, а програмне забезпечення. Elgato надає набір засобів для розробки (SDK), через який сторонні розробники зробили плагіни майже для всього: OBS, Streamlabs, Spotify, Twitch, Discord, Philips Hue, Home Assistant, навіть для Adobe Photoshop і AutoCAD. Окрема

категорія плагінів - Multi Actions, які перетворюють Stream Deck на повноцінний макропад: одна кнопка послідовно виконує кілька команд із заданими затримками [5]. На офіційному каталозі плагінів зараз доступно понад тисячу розширень, причому переважна більшість з них - безкоштовні. Окремі сторонні розробники роблять платні плагіни для специфічних сценаріїв на кшталт інтеграції з системами «розумного будинку» чи з конкретними інженерними пакетами, але це радше виняток.

З погляду конкретних робочих сценаріїв Stream Deck зазвичай використовують у трьох основних ситуаціях. Перші, стрімери та подкастери, перемикають сцени, активують звукові ефекти, керують ботами у Twitch-чаті, викликають автоматичні поздоровлення для нових глядачів. Другі, відеоредактори та звукорежисери, прив'язують до пристрою конкретні дії у Premiere, Final Cut, DaVinci Resolve, Pro Tools. У цьому сценарії іконку на кнопці часто роблять схожою на іконку самої дії в інтерфейсі програми, що швидко формує м'язову пам'ять. Третя категорія, програмісти й системні адміністратори, запускає збірки, тести, перемикається між гілками Git, відкриває конкретні сторінки документації. Деякі компанії взагалі видають Stream Deck співробітникам як штатний інструмент для роботи з внутрішніми сервісами.

Лінійка моделей, актуальних на момент написання роботи, наведена у таблиці 1.1.

З таблиці 1.1 одразу видно одну річ: серед усієї лінійки лише одна модель - Stream Deck + - має енкодери, причому всього чотири. Для сценарію з керуванням гучністю восьми додатків її замало. Решта моделей це чисті макропади без жодної обертової ручки, тобто гучність ними можна змінювати лише дискретно (натиснув і стало на 5 % гучніше). Окрема проблема - програмне забезпечення Stream Deck є закритим і офіційно працює тільки під Windows та macOS; підтримки Linux немає, хоча існує сторонній відкритий проєкт streamdeck-ui [6].

Таблиця 1.1 - Моделі сімейства Elgato Stream Deck

Модель	**LCD-кнопок**	**Поворотні регулятори**	**Орієнтовна ціна, USD**
Stream Deck Mini	6	-	60–80
Stream Deck MK.2	15	-	130–160
Stream Deck XL	32	-	220–270
Stream Deck +	8	4 енкодери + сенсорна смуга	180–210
Stream Deck Neo	8	2 сенсорні зони	80–100

Друга важлива особливість - ціна. Найдешевша модель Stream Deck Mini з шістьма кнопками коштує близько 60–80 доларів, що приблизно дорівнює середній денній зарплаті в українському секторі IT. Топова модель Stream Deck XL з 32 кнопками коштує понад 220 доларів - це вже відчутна інвестиція навіть для професіонала. Враховуючи, що пристрій виконує допоміжну функцію і не є необхідним для роботи, ціновий бар'єр відсікає більшість потенційних користувачів. Зрозуміло, що для професіонала, який щодня виходить у прямий ефір, інвестиція швидко окупається. Але для звичайного користувача, який хоче просто зручніше керувати гучністю Spotify і Zoom, така ціна не виправдана.

1.3.2 TC-Helicon GoXLR

GoXLR - апаратний аудіомікшер для стрімерів, представлений у 2019 році компанією TC-Helicon (підрозділ Music Tribe, до якої також належить Behringer). Це найвідоміший пристрій, у якому фізичні фейдери поєднано з програмною маршрутизацією звуку на рівні операційної системи [8]. Загальний вигляд пристрою показано на рисунку 1.4.

Корпус GoXLR - це невеликий настільний пульт з чотирма моторизованими фейдерами зверху і блоком кнопок-семплпадів та голосових ефектів у нижній частині. Окремо є вхід XLR для конденсаторного мікрофона

з фантомним живленням +48 В, лінійний вхід, навушниковий вихід і два виходи USB на ПК. Усередині - окремий процесор обробки сигналу, який у реальному часі обробляє звук з мікрофона (компресор, де-есер, гейт, еквалайзер) і додає голосові ефекти: зміну висоти голосу, реверберацію, ехо, «мегафон», «робот-голос». Модель GoXLR Mini простіша: ті ж чотири фейдери, але без моторизації, без сампл-падів і без голосових ефектів.

Рисунок 1.4 - Апаратний аудіомікшер GoXLR

Моторизованість фейдерів - окрема цікава деталь. Це означає, що повзунки на пристрої можуть рухатися самі: коли програма змінює гучність каналу (наприклад, ведучий стрімінгу перемкнув сцену і записаний у сцені рівень гучності музики автоматично став іншим), фейдер фізично переїжджає у нове положення. Користувач не мусить пам'ятати, в якому стані канал зараз; достатньо подивитися на пристрій. Це запозичення зі студійних мікшерних консолей, де моторизовані фейдери Mackie Control чи аналогічних рішень коштують значно дорожче.

Найцікавіше у GoXLR не залізо, а спосіб інтеграції з ПК. При підключенні драйвер створює в Windows чотири окремих віртуальних аудіопристрої: Chat, Music, Game і System. Користувач у налаштуваннях Windows направляє кожен додаток у потрібний канал, наприклад, Discord у Chat, Spotify у Music, гру в Game. Чотири фізичні фейдери на пристрої

прив'язані саме до цих чотирьох каналів. Зрушив фейдер, змінив рівень відповідної шини, не торкаючись миші. На виході внутрішній процесор обробки сигналу змішує всі канали разом з обробленим мікрофоном і віддає одну стерео-доріжку слухачам трансляції, а паралельно окрему доріжку в навушники самого стрімера, де можна, наприклад, зробити «свою» музику тихішою для аудиторії, але залишити голосною для себе.

Окремі доріжки «для себе» та «для аудиторії» це студійна ідея моніторного сигналу, тільки в побутовому форматі. У професійних мікшерах для цього використовується кілька шин (main, monitor, aux); у GoXLR реалізовано спрощений варіант лише з двома. Цього вистачає для типового стрімера, але професійному звукорежисеру буде замало.

GoXLR - приклад того, як добре зроблене залізо застрягло в долі свого програмного забезпечення. Програма GoXLR App працює виключно під Windows, виключно з онлайн-активацією, і у 2023 році TC-Helicon фактично припинила розвиток продукту: припинились оновлення, кілька разів обривалась робота серверів активації, офіційні канали зв'язку перестали відповідати [9]. У відповідь спільнота розпочала проєкт goxlr-utility - відкриту реалізацію драйвера мовою Rust, яка не залежить від серверів TC-Helicon і дозволяє користуватися пристроєм під Linux та macOS [10]. Це показове: пристрій ціною 300–400 доларів може втратити функціональність через рішення однієї компанії, і єдиним способом його врятувати залишається відкрите програмне забезпечення. Окрема проблема - навіть у роботопридатному стані GoXLR жорстко обмежений чотирма каналами. Якщо одночасно треба окремо керувати Discord, Spotify, грою, OBS, Zoom і браузером - каналів просто не вистачить.

Цей випадок добре ілюструє, наскільки важливо для апаратного пристрою мати відкритий програмний бік. У ситуації, коли виробник зник, а пристрій залишився, користувач опиняється у пастці: фізичне залізо в нього є, а ввімкнути його не може, бо у драйвера прострочена ліцензія. Це і є непрозора

залежність від виробника, якої я свідомо уникаю у власній розробці. Мікропрограма мого пристрою публікується разом із вихідним кодом, що дозволяє будь-якій людині продовжити її розвиток навіть якщо я особисто покину проєкт.

1.3.3 Проєкт deej

Deej - відкритий апаратно-програмний проєкт, започаткований розробником Omri Harel у 2018 році й розміщений на GitHub [3]. Це найвідоміша саморобна альтернатива комерційним мікшерам: за 5–15 доларів можна зібрати фізичний мікшер гучності з кількома лінійними потенціометрами на корпусі. Типову реалізацію наведено на рисунку 1.5.

Рисунок 1.5 - Типова реалізація проєкту deej

Принцип роботи простий. Кілька потенціометрів (зазвичай від трьох до п'яти) підключаються до аналогових входів плати Arduino. Мікропрограма з фіксованою періодичністю опитує входи і шле значення в комп'ютер через послідовний порт у вигляді текстових рядків. Невеликий додаток мовою Go, що запускається на ПК, читає ці значення і встановлює гучність прив'язаних додатків через Windows Audio Session API (на Linux - через PulseAudio чи

PipeWire). Конфігурація прив'язки повзунок-додаток зберігається у звичайному текстовому файлі формату YAML. Завдяки відкритій ліцензії MIT спільнота створила десятки модифікацій: корпуси під 3D-друк, варіанти з більшою кількістю повзунків, інтеграцію з Discord та OBS.

Окремо варто згадати про спільноту deej. На офіційній сторінці проєкту опубліковано добірку саморобних реалізацій від користувачів з різних куточків світу, де люди показують свої корпуси, розводки, варіанти підсвічування. У цій спільноті обмінюються файлами для 3D-друку, схемами друкованих плат, варіантами мікропрограми, додатковими функціями (на кшталт LED-індикаторів рівня поточної гучності). Це характерний приклад того, як відкритий проєкт із простою основою породжує велику кількість похідних варіантів, кожен з яких адаптований під свого користувача.

Недоліки deej напряму випливають із його простоти. У базовому проєкті немає кнопок для макросів, немає підтримки енкодерів (лише лінійні потенціометри з обмеженим ходом), немає дисплея, немає вбудованої логіки макросів. Мікропрограма не використовує інтерфейс USB HID, тобто пристрій не може сам надсилати клавіатурні комбінації - усі дії виконує настільний додаток. Це робить deej хорошим стартовим прикладом, але далеким від повноцінного контролера у моєму сценарії використання.

1.4 Порівняльний аналіз і обґрунтування власного рішення

Зведений порівняльний аналіз розглянутих пристроїв за п'ятьма ключовими критеріями подано в таблиці 1.2. До неї додано колонку «Розроблюваний пристрій», щоб одразу зафіксувати цільові характеристики моєї роботи.

З таблиці 1.2 видно три речі. По-перше, Stream Deck закриває макроси, але поворотних регуляторів у нього мало навіть у топовій моделі, і навіть та модель коштує близько 200 доларів. По-друге, GoXLR закриває звукову частину, але обмежений чотирма каналами й залежить від програмного

забезпечення, яке виробник більше не розвиває. По-третє, deej дешевий і відкритий, але це чисті повзунки - без кнопок, без енкодерів, без можливості робити макроси.

Таблиця 1.2 - Порівняння розглянутих пристроїв із розроблюваним

Критерій	**Stream Deck +**	**GoXLR**	**deej**	**Розроблюваний пристрій**
Енкодерів (≥8)	4	0	0	8
Програмованих кнопок (≥12)	8 LCD	4 сампл-пади	0	12
Керування гучністю додатків	через плагін	4 фіксовані канали	прямо через WASAPI	прямо через WASAPI
Відкритий код мікропрограми й клієнта	ні	ні	так	так
Орієнтовна вартість, USD	180–210	300–400	5–15	~30

Звідси - список вимог до пристрою, який буде розроблено у цій роботі:

1. Не менше восьми поворотних енкодерів - щоб одночасно мати окрему ручку для гучності кожного з основних додатків (голосовий чат, музика, гра або відео, мікрофон, браузер, програма для роботи зі звуком тощо) без перемикання сторінок.
2. Не менше дванадцяти програмованих кнопок - для виклику макросів (вимкнення мікрофона, перемикання сцени, гарячі клавіші професійного ПЗ).
3. Керування гучністю окремих додатків операційної системи напряму, а не лише чотирьох фіксованих віртуальних каналів.

4. Відкрита мікропрограма й відкритий настільний клієнт, щоб не потрапити в ситуацію GoXLR, коли пристрій стає некорисним після відмови виробника.
5. Орієнтовна сумарна вартість компонентів - близько 30 доларів, тобто на порядок менша за вартість найдешевшого комерційного аналога з порівнянною функціональністю.

Розглянемо кожну з цих вимог докладніше, щоб обґрунтувати, звідки взялися саме такі числа і чому вони мають значення.

Типовий цільовий користувач має близько восьми одночасно активних аудіопотоків: голосовий чат (Discord чи Zoom), фонову музику (Spotify), браузер з відео (YouTube, Netflix), мікрофон, системні звуки, гру, програму для запису плюс один-два запасні канали. Звідси й число вісім. Менша кількість регуляторів — наприклад, чотири, як у Stream Deck + — змушувала б постійно перемикатися між сторінками або взагалі відмовлятися від частини каналів. Більша (дванадцять чи шістнадцять) ускладнює корпус, збільшує ціну і розштовхує регулятори далі одне від одного, що псує ергономіку.

Дванадцять кнопок випливають з типового набору гарячих макросів стрімера або відеоредактора: вимкнути мікрофон, вимкнути камеру, перемкнути сцени 1–4, запустити запис, зупинити запис, пауза та ще чотири довільні дії користувача. Зручно розмістити їх матрицею 3×4 — це компактний прямокутний блок під енкодерами, який не з'їдає багато площі. Восьми було б замало для базових функцій, а шістнадцять виводять корпус за межі настільного формату.

Підхід GoXLR з фіксованими віртуальними пристроями має принципову ваду: він жорстко прив'язує конкретний фейдер до конкретної шини. Закрив гру, відкрив відеоредактор — і фейдер «Game» уже нічим не керує. У моєму проєкті обрано інший підхід: програма на стороні ПК отримує події з пристрою і динамічно призначає їх живим аудіосесіям через WASAPI у Windows (або PulseAudio чи PipeWire у Linux). Ручка прив'язується до

конкретного процесу за іменем — наприклад, до spotify.exe — і скільки б разів користувач не закривав і не відкривав Spotify, вона продовжує керувати саме ним.

Аргумент проти закритого програмного забезпечення вже наведений на прикладі GoXLR: коли виробник зникає, разом з ним зникає і функціональність пристрою. Для саморобки на основі Arduino цей аргумент має ще одну грань: людина, яка зібрала пристрій своїми руками, очікує можливості його доробити. Якщо мікропрограму нікому модифікувати, користувач не зможе ні додати власну функцію, ні виправити баг, ні портувати її на іншу плату. У моєму проєкті мікропрограма і супутня програма опубліковані на GitHub, тож будь-хто може використовувати або переробити їх під свої потреби.

Цифра 30 USD з'явилася після порівняння цін на основні компоненти на типових майданчиках (AliExpress, LCSC, локальні крамниці у Києві). Орієнтовний кошторис такий: плата мікроконтролера на ATmega32U4 — 5–8 USD, вісім поворотних енкодерів з кнопкою — близько 7–10 USD (у середньому 1 USD за штуку), дванадцять тактових кнопок — 2 USD, три зсувних регістри 74HC165 — 1 USD, друкована плата — 3–5 USD, пластик для 3D-друку корпусу — 2–3 USD, дроти, рознімачі та дрібна електроніка — ще 2–3 USD. Виходить 22–32 долари. На цьому фоні Stream Deck Mini за 60 USD з його шістьма кнопками й без жодного енкодера виглядає неконкурентоздатним.

Жоден з трьох розглянутих пристроїв одночасно цим п'яти вимогам не відповідає. Stream Deck виконує лише пункт 2 (і то частково). GoXLR - пункт 3 (і то частково, з обмеженням у 4 канали). Deej - пункт 4 і близько до пункту 5, але не закриває пункти 1, 2 і 3. Звідси й береться ідея власного пристрою: взяти макроси від Stream Deck, прив'язку до системного аудіо в стилі GoXLR і відкритість deej, але в одному корпусі.

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі я розбирався, навіщо взагалі потрібен окремий апаратний контролер для ПК і чому з тим, що вже продається, мені не по дорозі.

Перше, з чого я почав, - стандартний мікшер Windows. Він робить рівно те, що мав робити у 2007 році: дозволяє виставити гучність кожній програмі мишею. Для сучасного сценарію, де одночасно йдуть голосовий дзвінок, фонова музика, запис екрана й чат у Discord, цього мало. Кожне коригування - це кліки в окремому вікні, що під час прямого ефіру виглядає так собі.

Далі я пройшовся по основних готових пристроях. Elgato Stream Deck закриває макросну частину дуже добре: дисплей під кнопками, зручне програмне забезпечення, велика спільнота розробників плагінів. Проте поворотних регуляторів у нього або зовсім немає, або їх кілька (у моделі Stream Deck +), і коштує він суттєво дорожче за решту. GoXLR навпаки, цікавий саме регуляторами, але обмежений чотирма каналами, повністю закритий, а його виробник, схоже, продукт уже не розвиває. Лишається deej, відкритий проєкт, фактично безкоштовний, але це чисто повзунки: ні кнопок, ні енкодерів, і тільки під Windows у базовому варіанті.

Якщо скласти все разом, виходить така картина: жоден з цих варіантів не дає одночасно вісім енкодерів, дванадцять кнопок, відкриту мікропрограму й ціну в районі 30 доларів. Кожен закриває свою половину задачі - або макроси, або гучності - і робить це під свою екосистему. Це створює прогалину, яку якраз і має закрити мій пристрій.

Окремий висновок, який я зробив для себе під час аналізу, стосується саме закритості програмного забезпечення. Випадок із GoXLR показав, що навіть успішний комерційний пристрій може стати непридатним до використання, якщо виробник вирішить припинити підтримку. Для довгострокового користування важливо, щоб і мікропрограма, і програма на ПК були відкритими - це знімає залежність від однієї компанії й дозволяє

спільноті продовжувати розвиток навіть тоді, коли початковий автор втрачає інтерес до проєкту.

Саме тому я й беруся розробляти власний пристрій. Він поєднає вісім поворотних енкодерів для одночасного керування гучністю кількох додатків, дванадцять програмованих кнопок для макросів, прямий зв'язок із системним звуковим рушієм замість фіксованих віртуальних каналів і повністю відкритий код мікропрограми та супутньої програми. Вартість компонентів планую утримати у межах 30 доларів, що на порядок дешевше за найдоступніший комерційний аналог із порівнянною функціональністю. Такий пристрій закриває всі п'ять вимог зі списку вище. І, що головне, не зав'язаний на одну компанію: якщо я колись зникну, мікропрограма й код залишаться, а інші люди зможуть продовжити.

РОЗДІЛ 2
РОЗРОБКА АПАРАТНОЇ ТА ПРОГРАМНОЇ ЧАСТИН КОНТРОЛЕРА КЕРУВАННЯ МУЛЬТИМЕДІА

2.1 Призначення та вимоги до пристрою

У цьому дипломному проєкті передбачається розробити, виготовити макетний зразок та написати програмне забезпечення для апаратного контролера, який бере на себе ту частину роботи з мультимедіа, де миша й клавіатура надто повільні: одночасне регулювання гучності кількох програм та виклик заздалегідь заданих макросів. Висновки першого розділу зафіксували розрив на ринку – жоден з розглянутих готових пристроїв не дає водночас велику кількість поворотних регуляторів, набір програмованих кнопок, відкритий код і низьку ціну. Саме під цей розрив і проєктувався пристрій.

Перелік функціональних вимог сформовано на основі порівняльного аналізу з підрозділу 1.4. Пристрій має забезпечувати:

1. Не менше восьми поворотних енкодерів, кожен з вбудованою кнопкою, щоб мати окрему ручку для гучності кожного основного аудіопотоку без перемикання сторінок.
2. Не менше дванадцяти програмованих тактильних кнопок для виклику макросів та гарячих клавіш.
3. Передачу подій на персональний комп'ютер через стандартний послідовний порт, а також здатність самостійно діяти як USB-клавіатура, медіапульт і геймпад, не потребуючи драйвера.
4. Інформаційний дисплей для відображення стану пристрою та поточного навантаження системи.

5. Живлення й обмін даними через єдиний кабель USB, без зовнішнього блока живлення.

6. Орієнтовну сумарну вартість компонентів у межах 30 доларів США.

Окремо висувалася вимога ремонтопридатності та повторюваності. Пристрій мав бути зібраний з компонентів, які легко придбати й перепаяти вручну, а не зі спеціалізованих мікросхем у дрібних корпусах. Цей критерій прямо вплинув на низку рішень: вибір через-отвірних (THT) енкодерів і кнопок, використання великих SMD-резисторів типорозміру 2010 замість мініатюрних, застосування мікросхем у корпусі DIP. Прилад замислювався як такий, що його зможе повторити людина із базовими навичками паяння, маючи на руках лише мікропрограму, файли плати та моделі корпусу.

Головна мета розробки – продемонструвати, що повноцінний контролер мультимедіа, який поєднує незалежне керування гучністю окремих застосунків, програмовані макроси та здатність працювати автономно без драйвера, можна побудувати на загальнодоступній елементній базі, утримати його вартість на порядок нижчою за комерційні аналоги і зробити повністю відкритим. Решта підрозділів послідовно показують, як цю мету було досягнуто на практиці.

2.2 Структурна схема пристрою

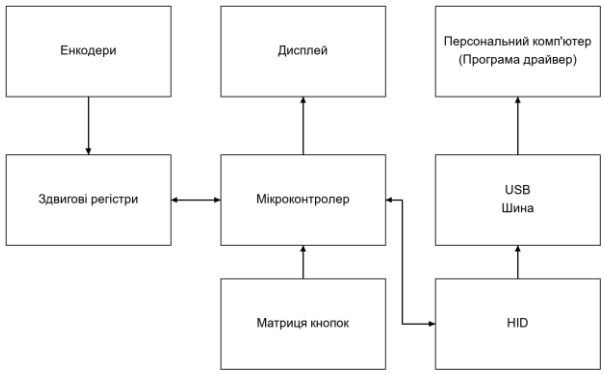


Рисунок 2.1 – Структурна схема пристрою

Розроблена структурна схема пристрою наведена на рисунку 2.1. Вона відображає логічну взаємодію між блоками пристрою: енкодерами та зсувними регістрами, матрицею кнопок, мікроконтролером, дисплеєм, інтерфейсом HID, шиною USB та програмою-драйвером на боці персонального комп'ютера.

Центральним елементом схеми є мікроконтролер, який одночасно виконує роль і обчислювального ядра, і апаратного інтерфейсу зв'язку з комп'ютером. Стан поворотних енкодерів зчитується не напряму, а через проміжний блок зсувних регістрів, які перетворюють велику кількість паралельних ліній у компактний послідовний потік. Кнопки організовані в матрицю та опитуються мікроконтролером безпосередньо. На дисплей мікроконтролер виводить інформацію про поточний режим і стан системи.

Стрілки на схемі показують напрямок інформаційних потоків. Від органів керування сигнали сходяться до мікроконтролера: енкодери віддають свій стан зсувним регістрам, регістри – мікроконтролеру, матриця кнопок надсилає сигнали напряму. Опрацювавши ці сигнали, мікроконтролер через блок HID та шину USB обмінюється даними з персональним комп'ютером, на якому працює програма-драйвер.

У бік комп'ютера пристрій формує два логічні канали. Через перший він представляється операційній системі складеним HID-пристроєм (клавіатура, пристрій керування медіа та геймпад) і може самостійно надсилати натискання клавіш та команди керування відтворенням, не потребуючи жодного встановленого програмного забезпечення. Другий канал – віртуальний послідовний порт, яким текстові події (поворот енкодера, натискання кнопки) передаються у програму-драйвер; вона вже на боці комп'ютера вирішує, що з цією подією зробити – змінити гучність конкретного застосунку, виконати гарячу клавішу чи запустити програму. Поділ відповідальності між «німим» пристроєм і «розумним» драйвером закладений у саму архітектуру і докладніше розкривається в підрозділі 2.8.

На основі структурної схеми формується перелік необхідних компонентів, кожен з яких розглядається в наступному підрозділі: мікроконтролерна плата, зсувні регістри, поворотні енкодери, кнопки та символьний дисплей, а також допоміжні елементи обв'язки – підтягувальні резистори й діоди матриці.

2.3 Вибір та обґрунтування елементної бази

2.3.1 Вибір мікроконтролерної плати

Робота всього пристрою впирається в одну ключову вимогу до мікроконтролера – апаратну підтримку USB HID. Пристрій має самостійно представлятися комп'ютеру як клавіатура й медіапульт, а це можливо лише тоді, коли інтерфейс USB реалізований апаратно в самому кристалі, а не емулюється програмно. Ця вимога одразу відсікає більшість популярних плат: Arduino Uno та Nano побудовані на ATmega328P, який не має вбудованого USB і спілкується з комп'ютером через окрему мікросхему-перетворювач, тому без повторного програмування допоміжного контролера прикидатися клавіатурою не вміє. Плати на ESP32 апаратний USB мають, але їхній рівень логіки 3,3 В ускладнив би узгодження з 5-вольтовими енкодерами та дисплеєм, а більшість їхніх можливостей (Wi-Fi, Bluetooth) для цього пристрою зайві.

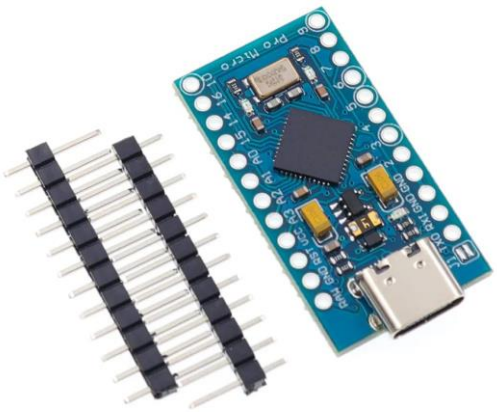


Рисунок 2.2 – Плата Arduino Pro Micro на базі ATmega32U4

Для реалізації проєкту було обрано плату Arduino Pro Micro з роз'ємом USB Type-C, побудовану на мікроконтролері ATmega32U4 (рис. 2.2). На відміну від ATmega328P, цей мікроконтролер має інтегрований повношвидкісний контролер USB, що й робить апаратний HID можливим «з коробки» [11].

Arduino Pro Micro – це компактна налагоджувальна плата, що поєднує мікроконтролер ATmega32U4, кварцовий резонатор, стабілізатор напруги та роз'єм USB на платі розміром 33×18 мм [12]. Основні технічні характеристики:

– мікроконтролер ATmega32U4, тактова частота 16 МГц, напруга живлення логіки 5 В;

– 32 КБ флеш-пам'яті (з яких близько 4 КБ зайнято завантажувачем), 2,5 КБ оперативної пам'яті SRAM та 1 КБ незалежної пам'яті EEPROM;

– вбудований повношвидкісний (12 Мбіт/с) контролер USB з підтримкою класів HID та CDC;

– близько 18 ліній цифрового вводу-виводу загального призначення, з них 5 з апаратним ШІМ та 9 із входами АЦП;

– роз'єм USB Type-C для живлення й даних.

Окрім USB-контролера, ATmega32U4 має чотири апаратні таймери, 12-канальний 10-розрядний АЦП, апаратний UART, апаратні інтерфейси SPI та I2C – у поточному пристрої більшість з них не задіяна, що залишає простір для розширення функціональності в наступних ревізіях. Програмування виконується через USB завдяки попередньо записаному завантажувачу Caterina: при подвійному короткому скиданні мікроконтролер на 8 секунд заходить у режим завантажувача й приймає нову мікропрограму без потреби у зовнішньому програматорі. Така схема програмування нічим не відрізняється від звичайного оновлення мікропрограми користувацького пристрою, тому процес доступний навіть людині без досвіду роботи з мікроконтролерами.

Енумерація пристрою в системі при під'єднанні відбувається стандартно для USB: мікроконтролер передає дескриптори пристрою та конфігурації, операційна система розпізнає його як складений HID-пристрій з кількома логічними інтерфейсами та автоматично завантажує штатні драйвери класів HID і CDC. Жодних спеціалізованих драйверів, файлів inf чи цифрових підписів встановлювати не потрібно.

Наявність 1 КБ EEPROM виявилася принциповою: саме в цій пам'яті пристрій зберігає конфігурацію макросів, тому в автономних режимах налаштування не втрачаються після від'єднання від комп'ютера. Підтримка одразу двох USB-класів – HID і CDC – дозволяє пристрою водночас і надсилати натискання клавіш, і обмінюватися текстовими командами з драйвером через віртуальний COM-порт, що лежить в основі всієї подальшої логіки [13].

На користь Arduino Pro Micro можна навести такі аргументи:

– апаратний USB HID без додаткових мікросхем і без повторного програмування це головна причина вибору;

– повна сумісність з екосистемою Arduino, через що доступні готові бібліотеки для роботи з USB, дисплеєм та зсувними регістрами;

– мала площа плати, що добре лягає в компактний настільний корпус;

– роз'єм Type-C, надійніший і зручніший за micro-USB старих ревізій;

– низька вартість (близько 5–8 доларів) при достатній для задачі обчислювальній потужності.

Отже, Arduino Pro Micro закриває критичну вимогу апаратного HID, лишається в межах бюджету і не тягне за собою додаткових компонентів обв'язки, що й зробило її основою пристрою.

2.3.2 Розширення кількості ліній вводу: зсувні регістри 74HC165

Восьмеро енкодерів дають 24 цифрові лінії (по дві фази та одна кнопка на кожен), і ще дванадцять кнопок матриці потребують своїх виводів. Навіть якщо матрицю згорнути до семи ліній, сумарна потреба перевищує число

вільних виводів Arduino Pro Micro. Постало завдання розширити кількість входів, не змінюючи мікроконтролер.

У ході аналізу було розглянуто чотири підходи до розширення цифрових входів, зведені у таблиці 2.1.

Таблиця 2.1 – Порівняння способів розширення кількості ліній вводу

Спосіб	**Зайнятих ліній МК**	**Інтерфейс**	**Особливості**
Прямі виводи МК	1 на кожен вхід	-	Виводів фізично не вистачає
Аналоговий мультиплексор 74HC4067	4 адресні + 1 сигнал	послідовне опитування	Читає по одному входу за раз, повільно для квадратури
Розширювач портів MCP23017	2 (I2C)	I2C	Залежить від тактування шини, складніша обробка
Зсувний регістр 74HC165	2 спільні + 1 на регістр	bit-bang SPI	Паралельне завантаження фіксує всі входи водночас

Вирішальним фактором стала природа сигналу з енкодерів. Квадратурний сигнал треба зчитувати дуже часто й бажано «миттєвим знімком» усіх ліній одразу, інакше при швидкому обертанні фази встигають змінитися між опитуваннями і крок губиться. Мультиплексор, який віддає по одному входу за раз, для цього підходить погано. Зсувний регістр 74HC165, навпаки, за командою паралельного завантаження фіксує стан усіх восьми своїх входів в один момент, після чого мікроконтролер спокійно вичитує цей знімок послідовно. Через це було обрано саме 74HC165 (рис. 2.3).

74HC165 – це 8-розрядний зсувний регістр із паралельним завантаженням і послідовним виходом (parallel-in, serial-out). Мікросхема має

вісім паралельних входів, вхід керування завантаженням, тактовий вхід і послідовний вихід; кілька регістрів можна об'єднувати каскадом для нарощування розрядності [14]. Основні характеристики:

– розрядність 8 біт, напруга живлення 2–6 В, сумісність з логікою 5 В;

– максимальна тактова частота до 25 МГц при 4,5 В (для цього проєкту вистачає кількох сотень кілогерц програмного тактування);

– корпус DIP-16, зручний для ручного паяння та заміни.

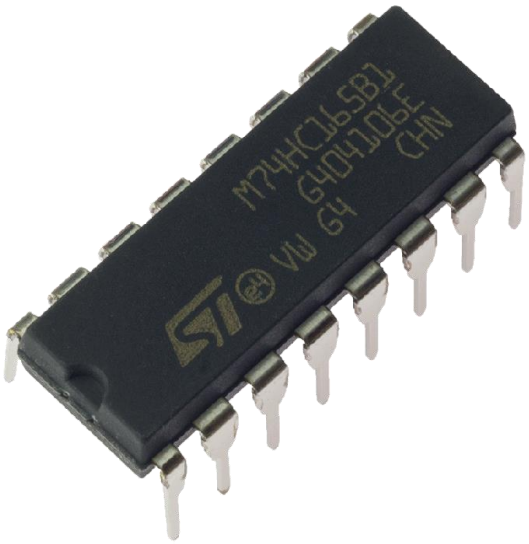


Рисунок 2.3 – Зсувний регістр 74HC165 у корпусі DIP-16

Принцип роботи 74HC165 базується на восьми внутрішніх D-тригерах, об'єднаних у каскад. На вхід керування завантаженням подається короткий низький імпульс – у цей момент усі вісім тригерів одночасно приймають значення зі своїх паралельних входів. Після зняття цього сигналу регістр переходить у режим зсуву: за кожним наростанням фронту тактового сигналу значення в тригерах зсувається на одну позицію в бік виходу QH, а на самому виході послідовно з'являються біти від старшого до молодшого. Окрім QH, мікросхема має ще й інверсний вихід (!QH), який у цьому проєкті не використовується. Для каскадного об'єднання кількох регістрів вихід QH першого зазвичай під'єднують до послідовного входу DS наступного – у такій конфігурації мікроконтролер бачить кілька регістрів як один довгий зсувний регістр. У цьому проєкті, як уже зазначалося, вибрано іншу схему – паралельне

читання трьох регістрів через окремі лінії QH, що дає втричі вищу пропускну здатність порівняно з каскадом.

Серед споріднених мікросхем розглядалися також регістри 74HC589 і 74HC597 з тристабільним вихідним буфером та вхідною засувкою, проте в цьому проєкті ці можливості надлишкові: лінії QH навантажені лише входами мікроконтролера, а захист від змін під час зчитування забезпечує спільний строб завантаження. Тому обрано саме базовий 74HC165 – найдешевший і найдоступніший варіант.

У пристрої застосовано три такі мікросхеми. Перший регістр збирає фази A всіх восьми енкодерів, другий – фази B, третій – лінії кнопок. Усі три регістри ділять спільні лінії завантаження та тактування, а їхні послідовні виходи заведені на три окремі виводи мікроконтролера, тому стан фаз A, B та кнопок зчитується одночасно, трьома бітами за один такт. Корпус DIP обрано свідомо: попри більший розмір порівняно з SMD-варіантом, його простіше монтувати вручну та, за потреби, замінити. Сумарна вартість трьох мікросхем не перевищує одного долара, що практично не впливає на бюджет.

2.3.3 Вибір поворотних енкодерів

Користувач крутить ручку гучності наосліп, не відриваючи погляду від основного екрана, тож головний орган керування пристрою має давати відчутний тактильний відгук і обертатися без обмежень. До поворотного регулятора висувалися конкретні вимоги: безкінечне обертання без крайніх положень, фіксовані позиції для оцінки «на дотик», наявність вбудованої кнопки та зручність монтажу. Цим вимогам відповідає інкрементальний механічний енкодер типу EC11 (рис. 2.4) – найпоширеніший формат у саморобній електроніці й комерційних аудіопультах.

Інкрементальний енкодер перетворює обертання вала у дві імпульсні послідовності, зсунуті по фазі на 90° (квадратурний сигнал). За тим, який канал випереджає інший, визначається напрямок обертання, а за кількістю імпульсів – величина повороту [15]. На відміну від потенціометра, енкодер

обертається безкінечно й не має механічних обмежувачів. Характеристики використаних енкодерів:

– 20 фіксованих позицій (клацань) на повний оберт та, як правило, стільки ж або вдвічі більше імпульсів;

– вбудована тактова кнопка, що спрацьовує при натисканні на вал;

– вал діаметром 6 мм (зазвичай зі зрізом під ручку-«ковпачок»);

– через-отвірний (THT) монтаж із механічними «вусиками» для фіксації на платі;

– ресурс обертання порядку 30 000 циклів [16].

Рисунок 2.4 – Інкрементальний поворотний енкодер типу EC11

Тактильні клацання тут виконують функціональну роль: під час регулювання гучності користувач за кількістю відчутих клацань приблизно оцінює, наскільки змінив значення, не дивлячись на екран. Вбудована кнопка дає кожному регулятору додаткову дію (наприклад, миттєве вимкнення звуку каналу), а через-отвірний монтаж і механічні вусики роблять енкодер стійким до бічних навантажень від пальців і простим у паянні. Усі вісім енкодерів – однакові, що спрощує і розведення плати, і обробку в мікропрограмі.

2.3.4 Підтягувальні резистори

Входи мікросхем 74HC165 не мають внутрішніх підтягувальних резисторів, тому в стані спокою їхній логічний рівень не визначений. Щоб кожна лінія енкодера мала чіткий рівень, її потрібно підтягнути до напруги

живлення. У пристрої застосовано схему з підтягуванням до +5 В: спільний вивід кожного енкодера з'єднано із землею, а кожну сигнальну лінію (дві фази й кнопку) підтягнуто резистором 10 кОм до +5 В. Завдяки цьому в спокої на лінії стабільна логічна одиниця, а під час замикання контакту вона притягується до нуля – тобто сигнал активний низьким рівнем (active-low).

На вісім енкодерів по три лінії припадає 24 підтягувальні резистори. Номінал 10 кОм є стандартним компромісом: достатньо великий, щоб струм через замкнений контакт був мізерним, і достатньо малий, щоб надійно утримувати рівень при наведених завадах. Резистори обрано у типорозмірі SMD 2010 (рис. 2.5).

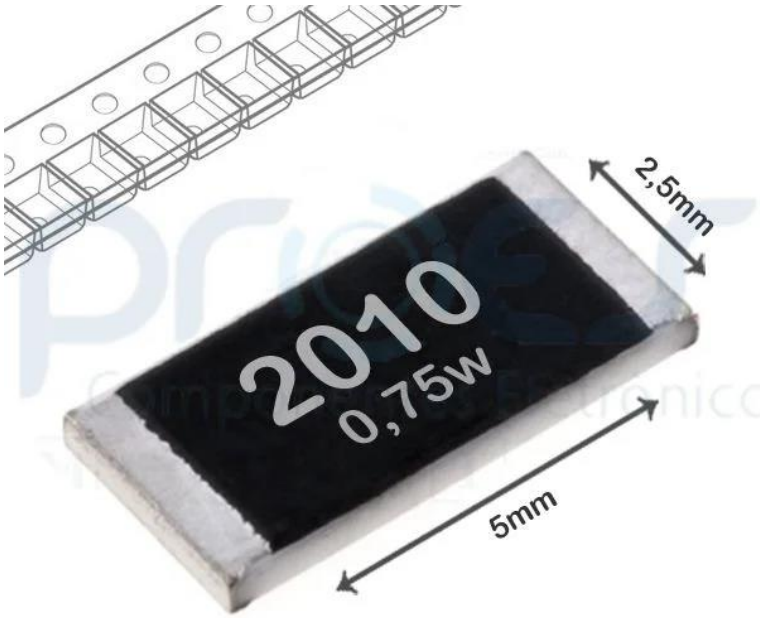


Рисунок 2.5 – Чип-резистор 10 кОм у типорозмірі 2010

Типорозмір 2010 (5,0×2,5 мм, метричне позначення 5025) – один з найбільших серед поширених чип-резисторів для поверхневого монтажу [17]. Для серійного виробництва такий розмір невигідний, але для ручного паяння він ідеальний: великі контактні майданчики легко лудяться звичайним паяльником, резистор не «тікає» з-під жала і його видно неозброєним оком.

Саме зручність ручного монтажу, а не електричні міркування, стала причиною вибору цього типорозміру – електрично резистор 10 кОм працює однаково в будь-якому корпусі. Це рішення прямо відповідає вимозі повторюваності пристрою, закладеній у підрозділі 2.1.

2.3.5 Матриця тактильних кнопок та діоди

Дванадцять програмованих кнопок призначені для макросів – від вимкнення мікрофона до перемикання сцен у програмі трансляції. Як клавіші обрано механічні перемикачі Cherry MX Red (рис. 2.6) – поширений стандарт серед механічних клавіатур, що забезпечує чіткий хід і великий ресурс.

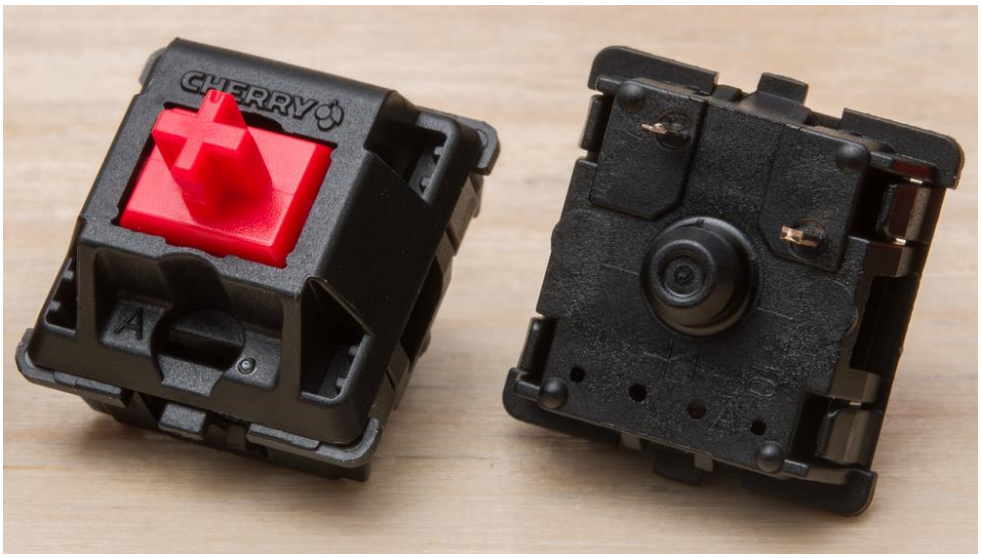


Рисунок 2.6 – Механічний перемикач Cherry MX Red

Cherry MX Red – лінійний перемикач без тактильного «горбика» й без клацання, з рівномірним зусиллям натискання впродовж усього ходу. Його характеристики [18]:

– зусилля спрацьовування близько 45 сН (45 г);
– точка спрацьовування на 2 мм, повний хід 4 мм;
– заявлений ресурс понад 100 млн натискань;
– сумісність зі стандартними клавішними ковпачками (keycap) з кріпленням «хрест».

У межах сімейства Cherry MX існують перемикачі з різним характером ходу – від лінійних без зворотного зв'язку до клацальних з виразним механічним «тиком». Зведений порівняльний огляд найпоширеніших варіантів і обґрунтування вибору Red для цього пристрою наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння поширених перемикачів Cherry MX

Варіант	**Характер ходу**	**Зусилля, сН**	**Шум**
MX Red	Лінійний	45	Тихий
MX Black	Лінійний	60	Тихий
MX Brown	Тактильний	45	Тихий зі слабким «горбиком»
MX Blue	Клацальний	50	Гучне клацання при спрацьовуванні
MX Silent Red	Лінійний	45	Дуже тихий

Для контролера макросів обрано саме лінійний варіант з невеликим зусиллям. Тактильний «горбик» MX Brown тут зайвий, бо користувачеві не потрібно відчувати точку спрацьовування – кнопка просто виконує дію. Клацальний MX Blue відсікається через сценарій використання: пристрій часто застосовується під час відеодзвінків і прямих ефірів, де гучне клацання потрапляє в мікрофон і дратує співрозмовників. MX Black з його збільшеним зусиллям 60 сН швидше втомлює пальці при повторюваних натисканнях, а MX Silent Red дорожчий за стандартний Red без значної переваги для настільного пристрою з відносно рідкими натисканнями. У сукупності цих чинників MX Red лишається найбільш практичним вибором – тихий лінійний хід за стандартну ціну.

Дванадцять кнопок зведено у матрицю 3×4 (три рядки на чотири стовпці), що дозволяє опитувати їх сімома лініями замість дванадцяти. Принцип сканування простий: мікроконтролер по черзі активує один стовпець і дивиться, які рядки відгукнулися. Проте проста матриця має відому ваду –

ефект «фантомних» (примарних) натискань: коли одночасно натиснуто три кнопки у формі літери «Г», струм знаходить обхідний шлях через сусідні замкнені контакти, і пристрій «бачить» неіснуюче четверте натискання [19].

Щоб уникнути цього, послідовно з кожною кнопкою встановлено діод, який пропускає струм лише в один бік і тим самим перекриває паразитні обхідні шляхи. Застосовано діоди 1N4148W у корпусі SOD-123 (рис. 2.7).

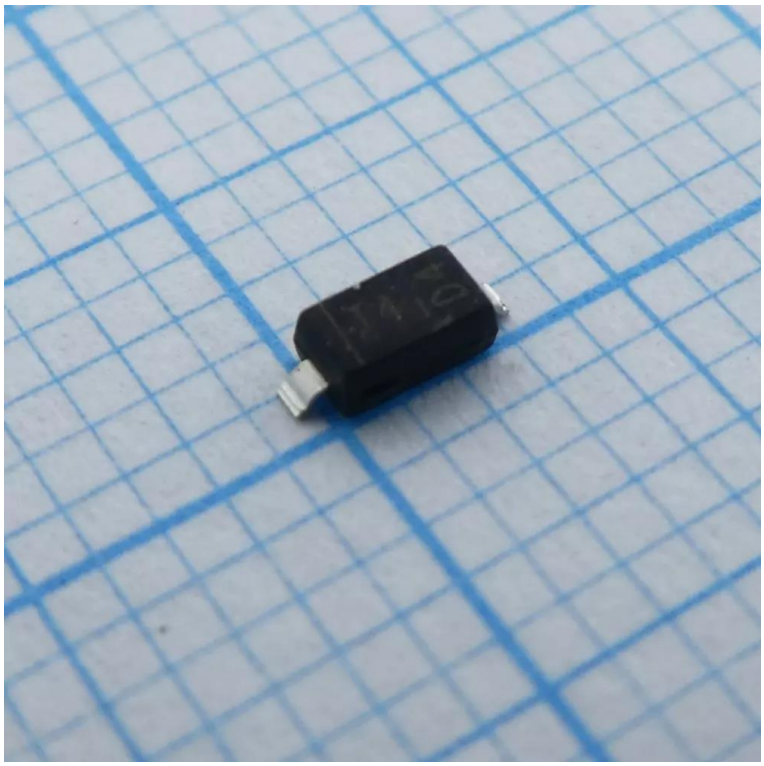


Рисунок 2.7 – Перемикальний діод 1N4148W у корпусі SOD-123

1N4148W – це швидкий кремнієвий перемикальний діод у малому SMD-корпусі SOD-123, з прямою напругою близько 0,7 В і допустимим постійним прямим струмом до 300 мА (середній випрямлений – 150 мА) [20]. Для матриці кнопок його електричні параметри з величезним запасом перекривають потреби – струм через лінію мікроскопічний. Цей діод обрано з іншої причини: 1N4148W у корпусі SOD-123 є фактичним стандартом у середовищі саморобних механічних клавіатур, тому він дешевий, повсюдно доступний і має готові посадкові місця в бібліотеках KiCad. Дотримання цього де-факто стандарту спрощує і закупівлю, і повторення пристрою іншими.

2.3.6 Символьний рідкокристалічний дисплей

Дисплей потрібен, щоб користувач бачив поточний режим пристрою та навантаження системи, не звертаючись до екрана комп'ютера. Для цього у пристрій додано символьний рідкокристалічний дисплей формату 20×4 (двадцять символів у чотири рядки) на поширеному контролері, сумісному з HD44780 (рис. 2.8). Класичне паралельне під'єднання такого дисплея вимагає від шести до десяти ліній мікроконтролера, що неприйнятно за дефіциту виводів. Тому застосовано модуль з припаяним перехідником на мікросхемі-розширювачі PCF8574, який перетворює дисплей на пристрій шини I2C.

Рисунок 2.8 – Символьний дисплей 20×4 з інтерфейсом I2C

I2C – це послідовна двопровідна шина (лінія даних SDA і лінія тактування SCL), до якої можна під'єднати кілька пристроїв, кожен зі своєю адресою [21]. Дисплейний модуль відгукується на адресу 0x27. Завдяки I2C усе під'єднання дисплея зводиться до двох сигнальних ліній плюс живлення, тобто звільняє щонайменше чотири виводи порівняно з паралельним інтерфейсом. Дисплей 20×4 вміщує 80 символів – цього достатньо, щоб

одночасно показувати завантаження процесора, пам'яті, відеокарти й диска, швидкість мережі та поточний режим. Підсвічування дозволяє читати показники в темному приміщенні, що типово для робочого місця стрімера чи відеоредактора [22].

2.3.7 Живлення пристрою

Пристрій живиться повністю від шини USB персонального комп'ютера: той самий кабель Type-C переносить і дані, і напругу +5 В. Окремого блока живлення чи акумулятора немає, що відповідає вимозі «один кабель» з підрозділу 2.1. Сумарне споживання залишається в межах стандартного ліміту USB-порту (500 мА для USB 2.0): мікроконтролер споживає одиниці міліампер, зсувні регістри й енкодери – частки міліампера, основним споживачем є підсвічування дисплея (десятки міліампер). Стабілізована напруга +5 В з USB подається на спільну шину живлення, від якої працюють мікроконтролер, три зсувні регістри, підтягувальні резистори та дисплей. Така схема живлення гранично проста й не потребує власних стабілізаторів.

2.3.8 Зведений перелік компонентів та орієнтовний кошторис

Підсумовуючи вибір елементної бази, у таблиці 2.3 наведено повний перелік компонентів пристрою з кількістю та орієнтовною вартістю. Ціни взято з типових майданчиків (AliExpress, LCSC, локальні крамниці) станом на час розробки й заокруглено.

Сумарна вартість компонентів становить близько 32 доларів, що лише незначно перевищує закладений у вимогах орієнтир у 30 доларів і залишається на порядок нижчою за вартість найдоступнішого комерційного аналога з порівнянною функціональністю. Перевищення пояснюється роздрібними цінами при купівлі компонентів поштучно: при замовленні деталей одразу на кілька пристроїв ціна одного комплекту опускається нижче за орієнтир. Найбільшу частку кошторису формують енкодери та плата мікроконтролера; решта компонентів коштують копійки й на загальну ціну майже не впливають.

Для порівняння: найдешевший Stream Deck Mini за 60–80 доларів не має жодного поворотного регулятора, а Stream Deck + із чотирма енкодерами коштує близько 200 доларів – у шість разів дорожче за цей кошторис.

Таблиця 2.3 – Орієнтовний кошторис компонентів пристрою

Компонент	**К-сть**	**Ціна за од., USD**	**Сума, USD**
Arduino Pro Micro (USB-C)	1	6,5	6,5
Поворотний енкодер EC11 з кнопкою	8	1,0	8,0
Перемикач Cherry MX Red	12	0,2	2,4
Зсувний регістр 74HC165 (DIP-16)	3	0,3	0,9
Резистор 10 кОм (SMD 2010)	24	0,02	0,5
Діод 1N4148W (SOD-123)	12	0,02	0,3
Дисплей LCD 20×4 з I2C-модулем	1	4,5	4,5
Друкована плата (2 шари)	1	4,0	4,0
Пластик PLA для корпусу	-	-	2,5
Дрібні матеріали (проводи, ковпачки, термовставки)	-	-	2,5
Разом			**≈ 32,1**

2.4 Розробка принципової електричної схеми

Принципову електричну схему пристрою було побудовано у вільному середовищі автоматизованого проєктування електроніки KiCad [23]. Повна схема наведена на рисунку 2.9. Нижче розглянуто, як з'єднані основні вузли – підсистема живлення, блок зчитування енкодерів, матриця кнопок і дисплей.

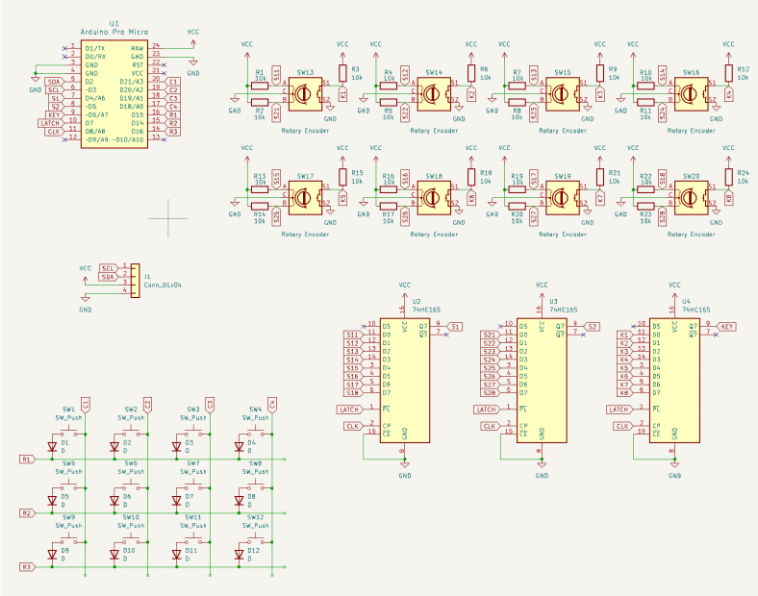


Рисунок 2.9 – Принципова електрична схема пристрою

Напруга +5 В надходить з виводу живлення плати Arduino Pro Micro, куди вона потрапляє безпосередньо з шини USB. Від цього вузла єдина шина +5 В розводиться на живлення трьох зсувних регістрів, спільні виводи підтягувальних резисторів і дисплейний модуль. Спільний символ GND також єдиний для всіх вузлів. Оскільки живлення береться вже стабілізованим з порту комп'ютера, а сигнали в схемі низькочастотні й передаються короткими доріжками всередині однієї невеликої плати, у прототипі окремі розв'язувальні (фільтрувальні) конденсатори біля мікросхем не встановлювалися; за потреби підвищення завадостійкості їх можна додати в наступних ревізіях плати.

До трьох регістрів 74HC165 заведено всі 24 сигнальні лінії восьми енкодерів. Кожен енкодер дає дві фази квадратурного сигналу (A і B) та кнопку під валом; спільний вивід з'єднано із землею. Сигнальні лінії підтягнуто резисторами 10 кОм до +5 В, тому в спокої на них тримається

логічна одиниця. Регістри не з'єднано каскадом: вони ділять спільні лінії завантаження (вивід D7 мікроконтролера) і тактування (D8), але кожен має власний послідовний вихід, заведений на окремий пін - D4 для регістра фаз A, D5 для регістра фаз B, D6 для регістра кнопок. За одну операцію завантаження мікроконтролер робить миттєвий «знімок» стану всіх 24 ліній, а потім за вісім тактів вичитує його, отримуючи по три біти за такт. Без такого паралельного зчитування при швидкому обертанні енкодера фази встигали б змінитися між опитуваннями, і крок було б втрачено.

Сканування дванадцяти кнопок виконується класичною матричною схемою 3×4. Чотири стовпці заведено на виводи-виходи мікроконтролера (аналогові входи A0–A3, що працюють тут як цифрові) і в спокої утримуються у високому стані; три рядки - на входи з увімкненим внутрішнім підтягуванням (виводи 14, 15 та 16). Сам цикл сканування простий: мікроконтролер притягує до низького рівня один стовпець, перевіряє стан рядків, повертає стовпець у високий стан і береться за наступний. Якщо на перетині активного стовпця й деякого рядка натиснута кнопка, відповідний рядок переходить в логічний нуль. Послідовно з кожною кнопкою стоїть діод 1N4148W - пропускає струм лише в один бік, чим виключає фантомні натискання при одночасному утриманні кількох клавіш.

Чотири проводи між платою і дисплеєм відповідають лініям апаратної шини I2C мікроконтролера: даних SDA на виводі D2, тактування SCL на D3, плюс живлення +5 В і земля. На відміну від решти компонентів, дисплей не впаяний у плату, а виведений на гнучкий джгут. Таке рішення продиктоване компонуванням корпусу: дисплей кріпиться до верхньої лицьової деталі, а плата затискається між половинами корпусу, тож кабель між ними дозволяє складати корпус без пайки в обмеженому просторі.

Повний розподіл задіяних виводів мікроконтролера між вузлами зведено у таблиці 2.4.

З огляду на швидкодію зчитування варто оцінити, скільки часу займає один повний опит енкодерів. Цикл складається з короткого імпульсу завантаження тривалістю кілька мікросекунд та восьми тактових імпульсів, після кожного з яких програма зчитує три біти. Усі ці операції реалізовано безпосередніми зверненнями до регістрів портів мікроконтролера, що скорочує тривалість одного імпульсу до одиниць мікросекунд (деталі програмної реалізації наведено в підрозділі 2.8.2). У сумі повний опит восьми енкодерів займає кілька десятків мікросекунд – значно швидше за найшвидше обертання вала пальцями, тому пропуск кроку через невчасне опитування виключений.

Таблиця 2.4 – Розподіл виводів мікроконтролера між вузлами

Вузол	**Сигнал**	**Вивід Pro Micro**
Зсувні регістри	Завантаження (LD)	D7
Зсувні регістри	Тактування (CLK)	D8
Зсувні регістри	Вихід регістра фаз A	D4
Зсувні регістри	Вихід регістра фаз B	D5
Зсувні регістри	Вихід регістра кнопок	D6
Матриця кнопок	Стовпці COL1–COL4	A0–A3 (18–21)
Матриця кнопок	Рядки ROW1–ROW3	15, 14, 16
Дисплей	SDA / SCL (I2C)	D2 / D3

Окремий момент стосується I2C-шини дисплея. Стандартна частота шини в проєкті задана у 100 кГц, що дає тривалість передачі одного байта близько 90 мкс. На повне оновлення рядка 20 символів припадає кілька мілісекунд, тому оновлення дисплея має значно нижчий пріоритет у головному циклі й розбите на окремі рядки – це описано в підрозділі 2.8.4.

Підсумовуючи, схема забезпечує узгоджену роботу всіх вузлів за єдиної логіки 5 В: живлення береться однією шиною з USB, 24 лінії енкодерів

зчитуються трьома паралельно з'єднаними регістрами, дванадцять захищених діодами кнопок опитуються матрично, а дисплей під'єднано двома сигнальними лініями. Кількість задіяних виводів мікроконтролера вкладається в наявний бюджет із запасом.

2.5 Розробка друкованої плати

Перш ніж замовляти виготовлення плати, прототип пристрою було розроблено на макетній платі зі скороченою кількістю енкодерів (рис. 2.10). Макет дозволив переконатися, що зчитування 74HC165 і квадратурне декодування працюють як задумано, і відпрацювати мікропрограму ще до того, як розведення буде зафіксовано в металі. Лише після цього схему перенесли в редактор плати.

Рисунок 2.10 – Прототип пристрою на макетній платі

Друкована плата (ДП) – це діелектрична основа з нанесеними мідними доріжками, яка механічно тримає компоненти й електрично з'єднує їх відповідно до принципової схеми, замінюючи навісний монтаж дротами. Розведення плати виконувалося в тому ж середовищі KiCad, що й схема, оскільки воно дозволяє передати схему безпосередньо в редактор плати зі збереженням усіх електричних зв'язків.

Маршрут проєктування плати був типовим для KiCad. Спочатку кожному компоненту схеми призначається посадкове місце (footprint) – для Arduino Pro Micro, енкодерів, кнопок, мікросхем DIP-16, чип-резисторів 2010 і діодів SOD-123. Потім схема переноситься в редактор плати у вигляді переліку з'єднань. Далі компоненти розставляються згідно з майбутньою ергономікою лицьової панелі: дисплей розташований у верхній частині, вісім енкодерів – у середній зоні, а блок кнопок – унизу, під пальцями. Після цього з'єднання розводяться доріжками. Завершальним кроком формуються виробничі файли (Gerber), за якими виготовляється плата. Готову розводку плати у редакторі KiCad наведено на рисунку 2.11.

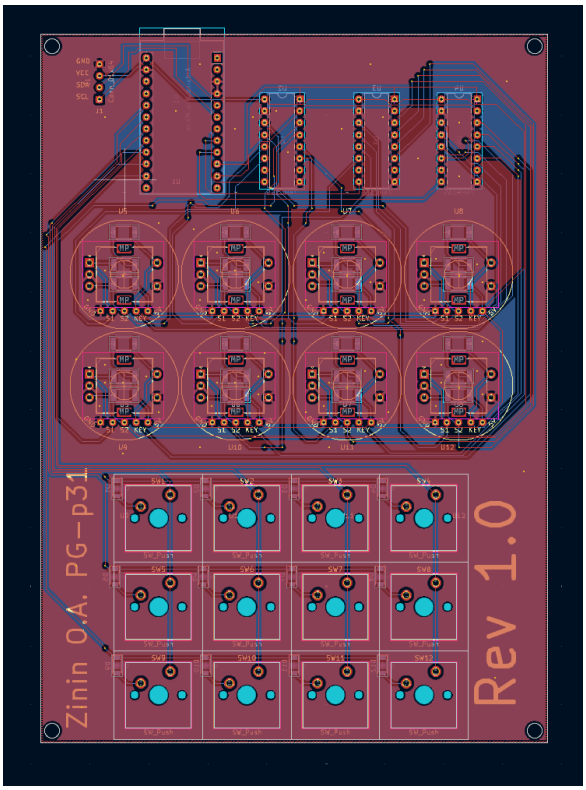


Рисунок 2.11 – Розводка друкованої плати у середовищі KiCad

Плата виконана у двошаровому варіанті: доріжки розведені по верхньому й нижньому боках, а вільні ділянки залиті полігоном спільного проводу. Заливка міддю виконує одразу дві задачі – спрощує розведення «землі», бо до неї можна під'єднатися майже в будь-якій точці, і зменшує наведені завади. Ширина доріжок і зазори між ними обрані 0,2 мм – із запасом, оскільки логічні струми в схемі малі, а напруга невисока. Розташування елементів підпорядковане ергономіці: вісім енкодерів розставлені сіткою 2×4 у середній зоні, під ними передбачені місця під підтягувальні резистори, дванадцять перемикачів утворюють блок матриці 3×4 і 12 діодів під ними, а три регістри і мікроконтролер - розміщені згори на зворотному боці плати. Усі компоненти, окрім дисплея, впаяні безпосередньо у плату без панельок і колодок; дисплей виведений на дроти. Виготовлену плату до монтажу компонентів показано на рисунку 2.12.

Рисунок 2.12 – Виготовлена друкована плата до монтажу компонентів

Готову плату було замовлено на типовому виробництві друкованих плат (Китай) у двошаровому виконанні. Замовлення промислового виготовлення замість саморобного травлення дало рівні доріжки, металізовані отвори, паяльну маску й шовкографію з позначеннями компонентів, що суттєво спростило подальший монтаж і зробило пристрій охайним.

Перед відправкою у виробництво виконано стандартну перевірку правил проєктування (Design Rule Check, DRC) у середовищі KiCad: вона автоматично виявляє типові помилки на кшталт занадто близько розташованих доріжок, неприєднаних виводів та перетину доріжок з обмежувальними областями. Після усунення зауважень DRC згенеровано виробничі файли формату Gerber – окремі шари верхньої та нижньої міді, верхньої та нижньої паяльної маски, верхньої шовкографії та таблиця координат отворів для свердління. Файл позиціонування компонентів (Centroid) експортовано додатково – на випадок, якщо в майбутніх ревізіях виникне потреба у машинному монтажі.

Сама плата виготовлена за класичним маршрутом фотолітографії: на двосторонній текстолітовій основі FR-4 з нанесеним з обох боків шаром міді експонується фоторезист через маски шарів, незахищена мідь стравлюється, отвори свердляться, металізуються (для з'єднання двох шарів), наноситься паяльна маска чорного кольору й шовкографія з позначеннями компонентів. Виводи відкритих контактних майданчиків покриті безсвинцевим припоєм методом гарячого лудіння HASL [39]. Усі ці процеси виконані на стороні виробника й не вимагали від розробника жодних додаткових операцій, окрім перевірки готового виробу візуально та мультиметром на коротке замикання шин живлення.

Поточна друкована плата є другою ревізією пристрою. Перша ревізія – описаний вище прототип на макетній платі – була зібрана на іншій платі сімейства Arduino з тим самим мікроконтролером, але з п'ятьма енкодерами замість восьми, і послужила пробою концепції: вона дозволила відпрацювати схемотехніку зчитування 74HC165, квадратурне декодування й роботу з I2C-дисплеєм. На основі досвіду першої ревізії в поточну версію закладено USB Type-C замість micro-USB, повний набір енкодерів, матрицю кнопок та перемикання між режимами роботи. Такий поетапний підхід дозволив виявити

вузькі місця мікропрограми на дешевшому макеті ще до того, як проєктування плати було завершено.

2.6 Конструювання корпусу

Корпус пристрою спроєктовано в системі автоматизованого проєктування SolidWorks [24] і виготовлено методом пошарового наплавлення пластикової нитки (FDM) на 3D-принтері з пластику PLA [25]. Корпус виконує дві задачі: фіксує плату й дисплей у визначеному взаємному положенні та формує лицьову панель, крізь яку виходять вали енкодерів і клавіші кнопок.

Конструкція складається з двох деталей – верхньої (лицьової) та нижньої. Верхня деталь несе на собі лицьову панель з отворами під вали восьми енкодерів, прорізами під дванадцять клавіш та вікном під дисплей. Нижня деталь утворює дно й бічні стінки. Загальний вигляд зібраної моделі та обидві деталі окремо показано на рисунках 2.13–2.15.

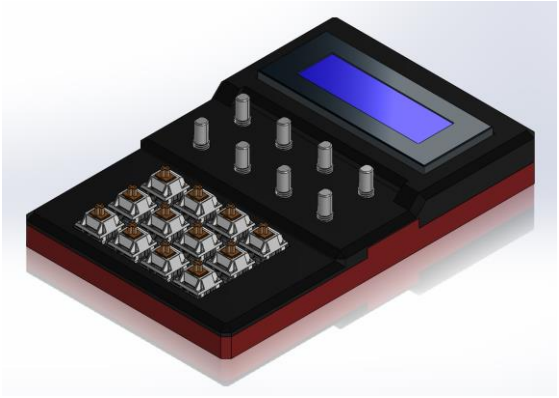


Рисунок 2.13 – Тривимірна модель корпусу пристрою

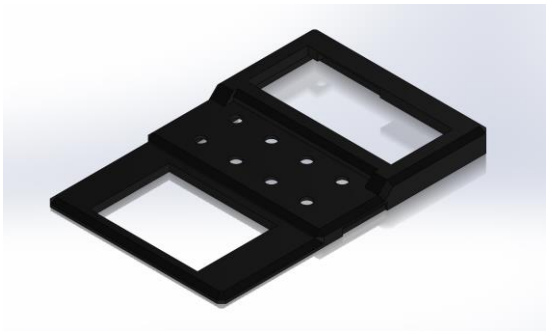


Рисунок 2.14 – Верхня частина корпусу з отворами під енкодери, кнопки та дисплей

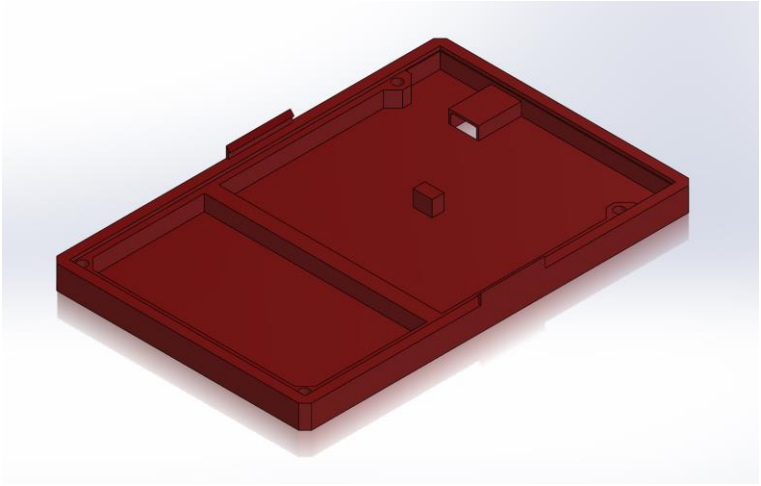


Рисунок 2.15 – Нижня частина корпусу

Дві половини корпусу з'єднуються між собою защіпками (snap-fit) без жодного гвинта: пружні виступи однієї деталі заходять у відповідні пази іншої. Друкована плата при цьому не кріпиться окремо, а просто затискається між верхньою й нижньою деталями по периметру, що спрощує і складання, і подальше обслуговування. Окремо кріпиться лише дисплей: він прикручений до верхньої деталі двома гвинтами М3, які вкручуються у термічні різьбові вставки, упресовані в пластик. Термовставки обрано тому, що різьба, нарізана прямо в PLA, швидко зношується при повторних розбираннях, тоді як латунна вставка дає надійне багаторазове різьбове з'єднання.

Роз'ємна конструкція із защіпками обрана з міркувань ремонтопридатності: пристрій можна багаторазово відкривати для доступу до плати, не маючи справи з окремим кріпленням самої плати, а відсутність гвинтів на корпусі робить його охайнішим. При моделюванні защіпок враховано пружність PLA та припуск на точність FDM-друку, щоб з'єднання трималося, але дозволяло розбирання без зламу виступів. Матеріал PLA обрано як найдоступніший і найпростіший у друку, чого для настільного пристрою, що не зазнає нагрівання чи механічних навантажень, цілком достатньо.

Параметри друку обрано з огляду на конструкційну роль деталей. Висоту шару задано 0,2 мм; для верхньої лицьової деталі, де якість видимої

поверхні має значення, верхні шари нанесено зі зменшеною швидкістю. Заповнення внутрішнього об'єму встановлено у 8 %. Товщина стінок 5 мм одночасно тримає корпус жорстким і залишає достатньо пружного матеріалу для роботи защіпок. Орієнтацію друку обрано так, щоб лицьова поверхня верхньої деталі формувалася з верхнього боку моделі, – це дає рівнішу поверхню без слідів підтримок.

Термічні різьбові вставки під дисплей встановлюються у відповідні отвори корпусу за допомогою нагрітого жала паяльника: тепло розплавляє навколишній пластик, вставка занурюється на потрібну глибину, пластик застигає і фіксує її.

Защіпки спроєктовано як консольні пружні елементи з невеликим зачіпним виступом. При складанні половин корпусу виступ заходить за паз протилежної деталі і фіксує її, а для розбирання достатньо невеликого зусилля на згин консолі. Розрахунковий хід защіпки в деформованому стані обрано в межах кількох десятих міліметра – менший за пружну деформацію PLA, тому защіпка повертається у вихідне положення без залишкової деформації.

2.7 Збірка пристрою

Збірка пристрою виконувалася поетапно, від найдрібніших поверхнево-монтованих елементів до великих через-отвірних компонентів і фінального складання в корпус.

Для монтажу використано стандартний набір інструментів: паяльник з регульованою температурою (рекомендована температура жала близько 320 °C для безсвинцевого припою), припій діаметром 0,5 мм з флюсом усередині, рідкий флюс для допомоги при паянні SMD-компонентів, тонкий пінцет, кусачки для через-отвірних виводів та плетений шнур-вбирач для усунення випадкових перемичок припою. Перевірку коротких замикань між шинами живлення і землі виконано звичайним мультиметром у режимі прозвонювання – обов'язковий етап перед першим під'єднанням плати до USB, щоб виключити пошкодження плати мікроконтролера через помилку монтажу.

На першому етапі монтувалися SMD-компоненти, доки плата ще порожня й до неї зручний доступ з усіх боків: 24 підтягуючі до живлення резистори типорозміру 2010 та діоди 1N4148W у корпусах SOD-123. Великі контактні майданчики резисторів 2010 дозволили виконати цей етап звичайним паяльником без паяльної пасти й фену.

Далі впаювалися три мікросхеми 74HC165 у корпусах DIP-16 з дотриманням орієнтації за ключем, а також плата Arduino Pro Micro. Наступним кроком встановлювалися через-отвірні компоненти лицьового боку – вісім поворотних енкодерів і дванадцять перемикачів Cherry MX Red; їхні механічні вусики й широкі виводи забезпечили жорстку фіксацію на платі. Повністю змонтовану плату показано на рисунку 2.16.

Рисунок 2.16 – Друкована плата з повністю змонтованими компонентами

Зібрану плату ще до встановлення в корпус підключали на столі для налагодження. Дисплейний модуль приєднувався чотирма проводами (живлення, земля, SDA, SCL), у мікроконтролер завантажувалася мікропрограма, і робота кожного органу керування перевірялася за подіями в моніторі послідовного порту й виводом на дисплей. Лише після того, як мікропрограма стабільно відпрацьовувала повороти й натискання, дисплей прикручувався до верхньої деталі корпусу через термічні вставки.

Завершується збірка встановленням плати між половинами корпусу: плата вкладається у нижню деталь, зверху накладається лицьова деталь з уже закріпленим дисплеєм, і обидві половини стискаються до клацання защіпок. На вали енкодерів надягаються ручки-ковпачки, на перемикачі - клавішні ковпачки. Зібраний пристрій показано на рисунку 2.17.

Рисунок 2.17 – Зібраний пристрій у корпусі

2.8 Розробка програмного забезпечення

2.8.1 Загальна архітектура програмного забезпечення

Програмне забезпечення пристрою складається з двох частин, що працюють на різних апаратних платформах. Перша – це мікропрограма (firmware) мовою C++, що виконується на мікроконтролері й відповідає за зчитування органів керування та формування подій. Друга – програма-драйвер мовою Python, що виконується на персональному комп'ютері й перетворює події на конкретні дії в операційній системі. Такий поділ дозволяє тримати мікропрограму простою й швидкою, а всю «розумну» логіку, що залежить від стану застосунків, винести на бік комп'ютера. Без цього поділу пристрій не зміг би, наприклад, прив'язати ручку до конкретної програми за іменем процесу – мікроконтролер не знає, які застосунки запущені, а драйвер знає.

Мікропрограму розроблено й зібрано у середовищі PlatformIO на базі фреймворку Arduino [26]. Як і будь-яка програма для Arduino, вона побудована за циклічною моделлю: одноразова функція ініціалізації `setup()` та функція `loop()`, що повторюється безкінечно. На етапі ініціалізації відбувається запуск послідовного порту, зчитування конфігурації з EEPROM, налаштування виводів, ініціалізація дисплея та реєстрація USB-класів HID. У головному циклі послідовно виконуються опитування енкодерів, обробка їхніх подій, сканування матриці кнопок, обробка команд із послідовного порту й оновлення дисплея.

PlatformIO обрано як середовище розробки замість штатного Arduino IDE з кількох причин. Воно зберігає всі залежності проєкту (бібліотеки з конкретними версіями, цільову плату, інструментарій компіляції) у єдиному файлі конфігурації platformio.ini, тому зібрати проєкт з нуля на іншому комп'ютері – справа однієї команди в терміналі. Стандартне Arduino IDE покладається на глобально встановлені бібліотеки, версії яких можуть розходитися між машинами й призводити до неочікуваних розбіжностей у поведінці мікропрограми. До того ж PlatformIO інтегрується з зовнішніми

редакторами коду та сучасними засобами навігації по проєкту, що суттєво прискорює налагодження.

Окрема технічна особливість мікропрограми – складений (composite) USB-пристрій. Мікроконтролер ATmega32U4 одночасно реєструє в системі чотири логічні пристрої: клавіатуру, пристрій керування медіа (Consumer Control), геймпад і віртуальний послідовний порт (CDC). Завдяки цьому операційна система бачить один фізичний пристрій як набір стандартних, для яких драйвери вже вбудовані, – встановлювати нічого не потрібно.

У мікропрограмі задіяно набір зовнішніх бібліотек, кожна зі своєю чіткою роллю:

– **HID-Project** [27] – реалізує складений USB HID-пристрій (клавіатура, Consumer, геймпад);

– **EncoderTool** [28] – декодує квадратурний сигнал енкодерів; її клас зчитування через 74HC165 використовується для опитування всіх восьми енкодерів;

– **Bounce2** [29] – усуває деренчання (bounce) контактів кнопок енкодерів;

– **LiquidCrystal_I2C** [30] – керує символьним дисплеєм через шину I2C;

– стандартні бібліотеки "Wire" (I2C) та "EEPROM" (незалежна пам'ять) фреймворку Arduino [31].

Програма-драйвер написана мовою Python із графічним інтерфейсом на бібліотеці Kivy [35] і використовує низку модулів для зв'язку з пристроєм та керування системою; їх розглянуто в підрозділі 2.8.5.

2.8.2 Опитування енкодерів і декодування обертання

Зчитування стану восьми енкодерів реалізовано через три зсувні регістри 74HC165 за принципом програмного інтерфейсу SPI (bit-bang). Спершу короткий імпульс на лінії завантаження фіксує («защіпує») у регістрах поточний стан усіх 24 ліній. Потім мікроконтролер вісім разів подає тактовий

імпульс і після кожного зчитує три послідовні виходи – таким чином за вісім тактів відновлюється повна картина фаз A, фаз B та кнопок усіх восьми енкодерів.

Безпосереднє декодування обертання виконує бібліотека EncoderTool за допомогою машини станів (state machine). Дві фази A і B утворюють дворозрядний код, який при справному обертанні змінюється за правилом коду Грея – за один крок змінюється лише один біт. Послідовність станів однозначно задає напрямок: один порядок проходження кодів відповідає обертанню в один бік, зворотний порядок – в інший. Перехід, за якого обидва біти змінюються одразу, вважається неможливим і трактується як деренчання контактів, тому відкидається. Часові діаграми фаз A і B при обертанні в обидва боки показано на рисунку 2.18, а логіку переходів узагальнено в таблиці 2.5.

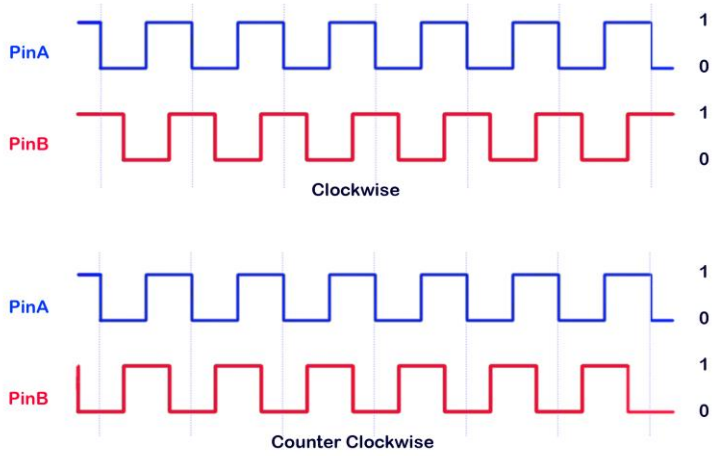


Рисунок 2.18 – Часові діаграми квадратурного сигналу енкодера

Таблиця 2.5 – Декодування квадратурного сигналу енкодера

Послідовність станів (A B)	**Інтерпретація**
00 → 01 → 11 → 10 → 00	обертання в один бік
00 → 10 → 11 → 01 → 00	обертання в інший бік
зміна обох бітів одночасно	хибний перехід, відкидається

У проєкті використано режим підрахунку "quarter", за якого один відчутний «клац» енкодера відповідає одній зміні значення лічильника. Ключова вимога до цього вузла – функцію опитування "tick()" потрібно викликати якомога частіше, бо при швидкому обертанні фази змінюються за частки мілісекунди, і рідкісне опитування призводить до пропуску кроків. Саме тому в головному циклі опитування й обробка енкодерів стоять найпершими і не «затиснуті» жодними затримками.

Дванадцять кнопок матриці опитуються окремо, з періодом 10 мс – для механічних кнопок такої частоти більш ніж достатньо. Алгоритм сканування по черзі активує кожен з чотирьох стовпців і зчитує три рядки, формуючи поточний стан усіх дванадцяти кнопок. Подія генерується лише тоді, коли стан кнопки відрізняється від попереднього, що автоматично відсіює утримання й уникає повторних спрацьовувань. Брязкіт контактів кнопок усередині енкодерів згладжує бібліотека Bounce2.

Для досягнення необхідної швидкості зчитування реалізацію bit-bang виконано безпосередніми зверненнями до апаратних регістрів портів мікроконтролера, без використання функцій digitalWrite та digitalRead фреймворку Arduino. Звичайний digitalWrite виконує перевірки на безпеку і займає десятки тактів процесора на одну зміну стану, тоді як пряма маніпуляція бітом регістра PORT – лише один такт. Завдяки цьому повний цикл опитування восьми енкодерів вкладається у час, помітно менший за тривалість найшвидшого клацання, і функцію tick() можна викликати кілька тисяч разів на секунду без помітного впливу на решту головного циклу.

Програмна реалізація замість апаратного блока SPI обрана тому, що мікроконтролер має лише один апаратний модуль SPI, орієнтований на одну послідовну лінію MISO. У цьому пристрої потрібно одночасно зчитувати три незалежні послідовні виходи (по одному з кожного регістра 74HC165), тож апаратний SPI вирішив би задачу лише частково – за кожним тактом усе одно довелося б додатково зчитувати ще два піни вручну. Bit-bang виявився

простішим і однорідним рішенням, оскільки три вихідні лінії регістрів обробляються однаково в межах одного циклу.

2.8.3 Режими роботи та обробка подій

Пристрій працює у трьох режимах, перемикання між якими здійснюється одночасним утриманням кнопок 11 і 12. Поточний режим зберігається в EEPROM, тому після повторного ввімкнення пристрій повертається в той самий режим.

– **Режим 1 (драйвер).** Пристрій лише надсилає текстові події в послідовний порт, а всі дії виконує програма-драйвер на комп'ютері. Це основний режим для керування гучністю окремих застосунків.

– **Режим 2 (HID-клавіатура/медіа).** Пристрій самостійно надсилає натискання клавіш, поєднання з модифікаторами та команди керування медіа згідно з конфігурацією, що зберігається в EEPROM. Драйвер у цьому режимі не потрібен.

– **Режим 3 (геймпад).** Кожен орган керування прямо відображається на кнопку віртуального геймпада (1–32): дванадцять кнопок, повороти енкодерів праворуч і ліворуч та натискання їхніх кнопок.

Незалежно від режиму кожна подія дублюється в послідовний порт у простому текстовому форматі, тому драйвер може «слухати» пристрій завжди. Протокол обміну з комп'ютером наведено в таблиці 2.6.

Конфігурація автономних режимів зберігається в EEPROM у вигляді 36 «дій», розподілених між органами керування за схемою з таблиці 2.7. Для режиму 2 кожна дія описує тип (клавіша чи медіа-команда), код і модифікатори; для режиму 3 – номер кнопки геймпада. Це дозволяє повністю переналаштувати поведінку пристрою через драйвер, не програмуючи його повторно.

На початку незалежної пам'яті зберігається контрольний байт-ознака (magic): якщо при ввімкненні він не збігається з очікуваним, мікропрограма

вважає пам'ять незаповненою й ініціалізує її нулями, що захищає від випадкових даних при першому запуску.

Таблиця 2.6 – Основні команди послідовного протоколу обміну

Напрямок	**Повідомлення**	**Призначення**
Пристрій → ПК	"E:B:<n>:<0/1>"	Натискання/відпускання кнопки 1–12
Пристрій → ПК	"E:R:<n>:<R/L>"	Поворот енкодера 1–8 праворуч/ліворуч
Пристрій → ПК	"E:P:<n>:<0/1>"	Натискання кнопки енкодера 1–8
Пристрій → ПК	"E:M:<m>"	Зміна режиму роботи
ПК → Пристрій	"SM:<m>"	Встановити режим
ПК → Пристрій	"CS:...", "CW"	Задати дію та записати конфігурацію в EEPROM
ПК → Пристрій	"SL:<cpu>:<ram>:<gpu>:<dsk>:<rx>:<tx>"	Передати навантаження системи для дисплея

Таблиця 2.7 – Розподіл 36 дій конфігурації за органами керування

Індекси дій	**Орган керування**
0–11	Кнопки 1–12
12–19	Енкодери 1–8, поворот праворуч
20–27	Енкодери 1–8, поворот ліворуч
28–35	Кнопки енкодерів 1–8

2.8.4 Алгоритм роботи мікропрограми

Узагальнений алгоритм роботи мікропрограми наведено на рисунку 2.19 у вигляді блок-схеми. Після ввімкнення виконується блок ініціалізації, далі програма входить у безкінечний головний цикл.

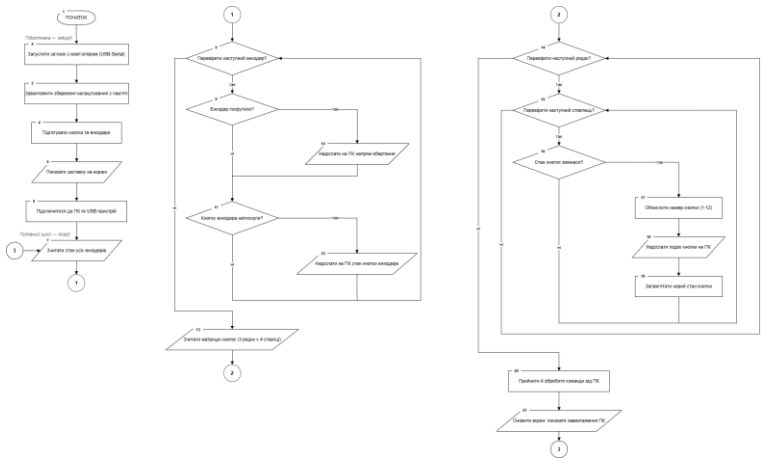


Рисунок 2.19 – Блок-схема алгоритму роботи мікропрограми

Послідовність кроків головного циклу побудована з урахуванням пріоритетів за швидкодією. Найперше й щоразу виконується опитування енкодерів та обробка їхніх подій – це найкритичніший до часу вузол. Далі за окремим таймером з періодом 10 мс виконуються менш термінові дії: сканування матриці кнопок, перевірка комбінації перемикання режиму та обробка натискань. Після цього обробляються команди з послідовного порту. Завершує цикл оновлення дисплея, яке відбувається не частіше ніж раз на 250 мс і записує лише один рядок за виклик, щоб звернення до повільної шини I2C не гальмувало решту циклу.

Щоб жодна операція не блокувала цикл надовго, застосовано кілька прийомів. Команди з послідовного порту читаються побайтово, без очікування цілого рядка. Оновлення дисплея розбите на окремі рядки з кешуванням, тому перемальовується тільки те, що змінилося. Формування імпульсів геймпада виконано неблокуюче – натискання й відпускання кнопки рознесені в часі через перевірку таймера, а не через затримку. Усі ці рішення підпорядковані одній меті – зберегти високу частоту опитування енкодерів, від якої залежить точність відстеження обертання.

За своєю природою такий підхід реалізує патерн кооперативної багатозадачності на однопотоковому мікроконтролері: завдання різного пріоритету (опитування енкодерів, сканування матриці, обмін з ПК, оновлення

дисплея) поділяють процесорний час, але кожне з них передає керування назад у головний цикл якомога швидше. На відміну від витісняльної багатозадачності, яка вимагала б операційної системи реального часу, тут планування відбувається на рівні поодиноких ділянок коду, а гарантії часу виконання забезпечує сам розробник через свідому відмову від блокуючих викликів. Для AVR без операційної системи реального часу це фактично єдиний практичний спосіб тримати швидкі енкодери на пріоритеті, не жертвуючи рештою задач.

2.8.5 Програма-драйвер на боці персонального комп'ютера

Програма-драйвер виконується на персональному комп'ютері й відповідає за роботу пристрою в основному режимі (режим 1), а також за налаштування автономних режимів і відображення навантаження системи. Вона побудована з кількох модулів, кожен зі своєю зоною відповідальності.

Модуль зв'язку з пристроєм відкриває віртуальний послідовний порт (через бібліотеку pySerial [36]), в окремому потоці читає текстові події від пристрою і викликає відповідні обробники. Реалізовано автоматичне під'єднання: програма приблизно раз на дві секунди переглядає доступні порти, знаходить серед них пристрій з ознаками Arduino й під'єднується до нього без втручання користувача; при від'єднанні кабелю програма це помічає й переходить у режим очікування.

Виконавчий модуль перетворює події на дії. Найважливіша з них – керування гучністю окремих застосунків: за поворотом енкодера драйвер через інтерфейс Windows Audio Session API (WASAPI) [32] знаходить аудіосесію потрібної програми за іменем її процесу й змінює рівень гучності кроком в один відсоток. Робота з аудіо реалізована через бібліотеку pycaw [33], що надає зручну обгортку над системними викликами Core Audio. Окрім гучності, дії можуть виконувати натискання клавіш (через бібліотеку keyboard [37]), команди керування медіа або запуск програм.

Окремий модуль моніторингу збирає системне навантаження для дисплея: завантаження процесора, оперативної пам'яті, диска й мережевий трафік через бібліотеку psutil [34], а завантаження відеокарти – через лічильники продуктивності Windows (Performance Counters, інтерфейс PDH бібліотеки pywin32 [38]), підсумовуючи активність усіх 3D-рушіїв, що працює незалежно від виробника відеокарти. Раз на секунду драйвер надсилає ці значення командою "SL:" на пристрій, де вони виводяться на дисплей.

Графічний інтерфейс програми реалізовано на бібліотеці Kivy. У вікні зліва зображена схема розташування енкодерів і кнопок, що підсвічує активний орган керування, а справа – панель налаштувань. Налаштування виглядає так: користувач обирає на діаграмі енкодер або кнопку, у панелі справа вказує тип дії (наприклад, «гучність застосунку»), а для гучності – ім'я процесу програми, якою керуватиме ця ручка. Налаштування режимів 2 і 3 драйвер передає на пристрій і записує в його EEPROM, налаштування режиму 1 зберігає локально у файлі формату JSON, тому вони не зникають між запусками. Загальний вигляд інтерфейсу наведено на рисунку 2.20.

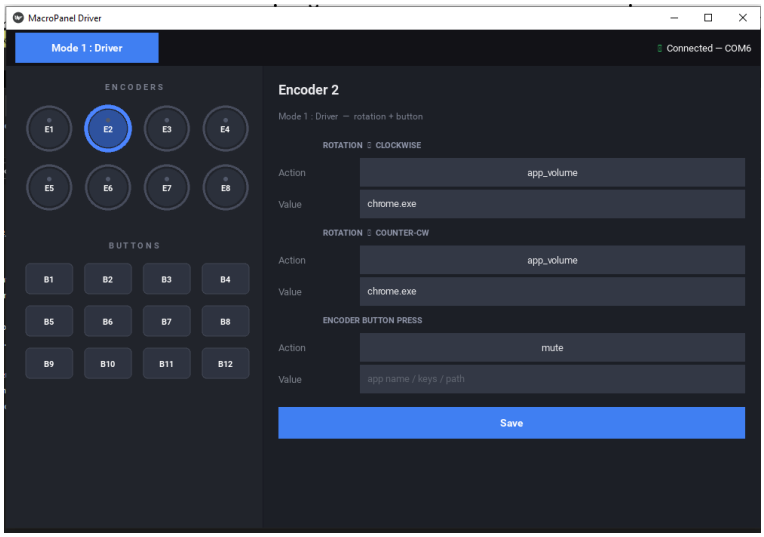


Рисунок 2.20 – Графічний інтерфейс програми-драйвера

Додати, наприклад, керування яскравістю монітора чи перемикання сторінок у браузері – це один новий обробник у виконавчому модулі; решта коду не торкається. Портування на іншу операційну систему вимагає

переписати лише модуль аудіо й, можливо, модуль моніторингу – модуль зв'язку з пристроєм та основна логіка інтерфейсу залишаться без змін, оскільки спираються на крос-платформні бібліотеки pySerial і Kivy.

2.8.6 Користувацький сценарій налаштування

Користувач під'єднує пристрій кабелем USB – і він одразу з'являється в системі як складений HID-пристрій та віртуальний COM-порт, без потреби у додаткових драйверах. Якщо встановлена програма-драйвер, вона автоматично знаходить пристрій і відкриває головне вікно з діаграмою органів керування.

Для основного режиму (режим 1) на діаграмі обирається потрібний орган керування – поворот енкодера праворуч або ліворуч, його кнопка чи будь-яка з дванадцяти матричних кнопок. У панелі праворуч з'являється набір параметрів: тип дії та її конкретні значення. Для регулювання гучності окремого застосунку обирається тип «гучність застосунку» й вказується ім'я процесу – наприклад, spotify.exe для музичного плеєра або chrome.exe для браузера. Для виклику комбінації клавіш обирається тип «клавіатура» і вказуються самі клавіші у форматі, що його розуміє бібліотека keyboard (наприклад, ctrl+alt+m для вимкнення мікрофона). Окрема дія «запуск програми» приймає шлях до виконуваного файлу і дає змогу прив'язати кнопку до швидкого запуску потрібного застосунку.

Налаштування автономних режимів (2 і 3) виконується у тих самих вікнах, але стосується дій, які пристрій виконує самостійно без участі драйвера. Для режиму 2 для кожного органу керування обирається стандартна клавіша (з повного списку HID-кодів USB), модифікатори (Ctrl/Alt/Shift/GUI) або медіа-команда (відтворення/пауза, наступний/попередній трек, регулювання гучності системою). Для режиму 3 кожен орган керування жорстко відображається на кнопку віртуального геймпада з номерами 1–32, тож додаткових налаштувань не потрібно. По завершенні редагування користувач натискає кнопку запису конфігурації, і всі параметри передаються

в незалежну пам'ять пристрою. З цього моменту пристрій працює в автономному режимі навіть на іншому комп'ютері без встановленого драйвера.

Перемикання між режимами виконується самим пристроєм за одночасним утриманням двох сусідніх кнопок матриці (11 і 12) протягом приблизно пів секунди. Поточний режим відображається в нижньому правому куті дисплея, тому користувач завжди бачить, у якому стані пристрій. Така самодостатня реалізація перемикання важлива тим, що дозволяє змінити режим навіть без під'єднання до комп'ютера або без запущеного драйвера.

2.9 Перевірка працездатності

Для перевірки працездатності пристрою було проведено ряд експериментів, що охопили як коректність основних функцій, так і поведінку мікропрограми під навантаженням. Найбільше клопоту на етапі налагодження завдала швидкодія, тому перший експеримент присвячено саме їй.

Експеримент 1. Швидкодія опитування енкодерів. На ранньому етапі при швидкому обертанні енкодери пропускали частину кроків. Заміри часу виконання операцій у головному циклі за допомогою функції `micros()` показали, що обробка послідовного порту блокувала цикл приблизно на 10 мс за виклик (`ser=10020` мкс). Через це функція опитування енкодерів викликалася лише близько 96 разів на секунду замість потрібних кількох тисяч, і квадратурний декодер не встигав за обертанням. Після переходу на неблокуюче побайтове читання команд частота опитування відновилася до кількох тисяч разів на секунду, і пропуск кроків при швидкому обертанні зник.

Експеримент 2. Затримка запису конфігурації. При записі повного набору з 36 дій режиму 2 графічний інтерфейс драйвера «зависав» на час до 18 секунд. Причиною виявилося те, що потік читання порту утримував блокування під час очікування рядка, через що кожна команда від інтерфейсу чекала. Після того як операцію читання було винесено з-під блокування, запис конфігурації став відбуватися практично миттєво.

Експеримент 3. Керування гучністю застосунку (режим 1). Енкодер було прив'язано до процесу музичного плеєра. Обертання праворуч і ліворуч плавно змінювало гучність саме цього застосунку кроком 1 %, що паралельно підтверджувалося рухом відповідного повзунка у системному мікшері гучності Windows. Гучність решти програм при цьому не змінювалася (рис. 2.21).

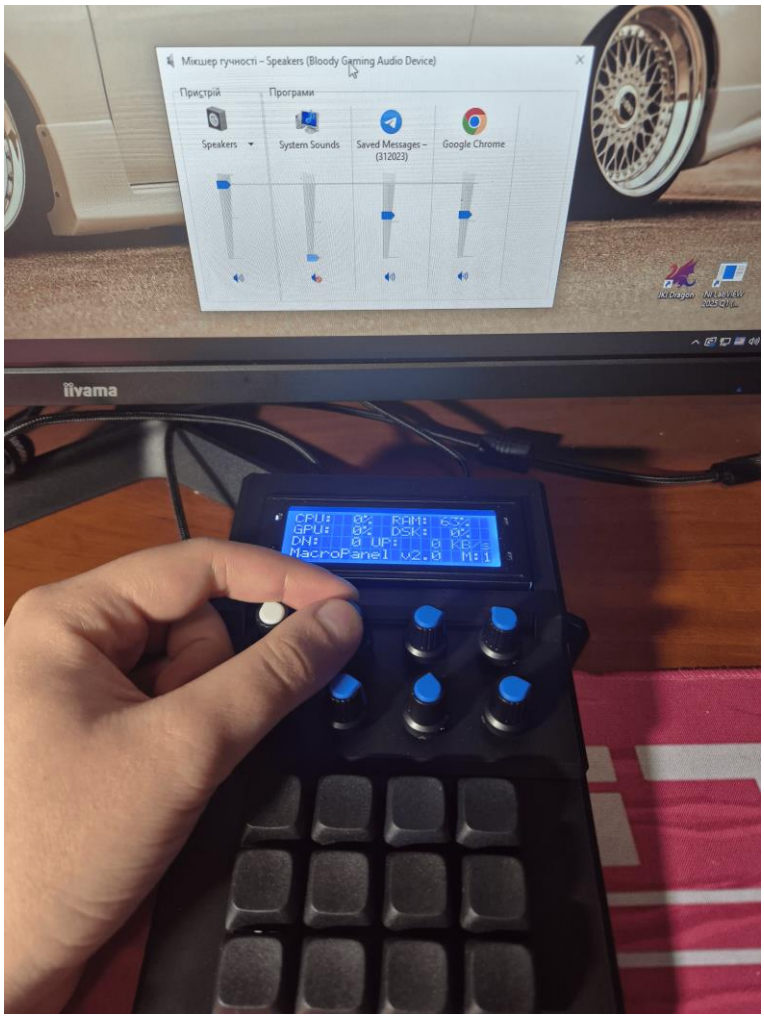


Рисунок 2.21 – Зміна гучності окремого застосунку в системному мікшері

Експеримент 4. Автономні макроси (режим 2). Кнопці було призначено поєднання клавіш для вимкнення мікрофона у програмі відеозв'язку. Натискання кнопки на пристрої, під'єднаному до комп'ютера без жодного драйвера, спрацьовувало як відповідне поєднання клавіш – пристрій коректно розпізнавався системою як USB-клавіатура.

Експеримент 5. Режим геймпада (режим 3). У вбудованому засобі Windows для перевірки ігрових контролерів натискання дванадцяти кнопок та повороти енкодерів відображалися як спрацьовування кнопок віртуального геймпада з номерами 1–32, тобто пряме відображення органів керування спрацювало правильно (рис. 2.22).

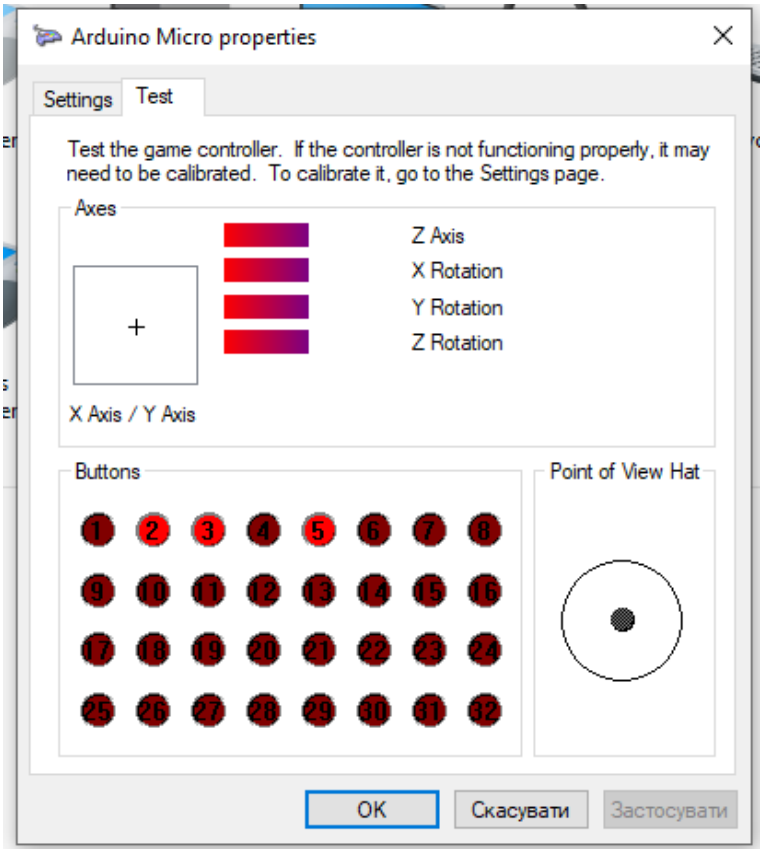


Рисунок 2.22 – Перевірка пристрою як геймпада засобами Windows

Експеримент 6. Перемикання та збереження режиму. Одночасне утримання кнопок 11 і 12 послідовно перемикало режими 1 → 2 → 3 → 1, що відображалося і на дисплеї, і повідомленням у послідовному порту. Після від'єднання й повторного під'єднання пристрою він повертався в той режим, що був активний останнім, – тобто стан режиму коректно зберігся в EEPROM.

Експеримент 7. Одночасна робота кількох органів керування. Перевірялося, чи не губляться події при одночасних діях. При обертанні двох енкодерів і утриманні кнопки водночас усі події надходили в послідовний порт

без втрат і без помилкових спрацьовувань сусідніх каналів – матриця з діодами та паралельне зчитування регістрів відпрацювали без збоїв.

Експеримент 8. Відображення навантаження на дисплеї. У режимі 1 при запущеному драйвері дисплей щосекунди оновлював показники завантаження процесора, пам'яті, відеокарти й диска та швидкість мережі (рис. 2.23). При від'єднанні драйвера приблизно через 3 секунди дисплей переходив у стан очікування з відповідним повідомленням, як і передбачено логікою обробки відсутності даних.

Рисунок 2.23 – Відображення навантаження системи на дисплеї пристрою

Експеримент 9. Тривала робота. Пристрій під'єднано до комп'ютера й залишено в активному стані з періодичним користуванням органами керування протягом восьми годин підряд. Жодних збоїв у роботі, втрат подій

або «зависань» мікропрограми не зафіксовано. Споживаний струм, виміряний за допомогою USB-перехідника зі вбудованим амперметром, утримувався стабільно на рівні близько 60–80 мА, причому основну частку формувало підсвічування дисплея, а сам пристрій у спокої практично не нагрівався. До ліміту USB-порту 500 мА залишається багаторазовий запас.

Проведені експерименти підтвердили, що прототип правильно виконує всі закладені функції: зчитує обертання восьми енкодерів без пропуску кроків, обробляє натискання дванадцяти кнопок, керує гучністю окремих застосунків, самостійно діє як HID-пристрій у двох автономних режимах і коректно відображає стан системи. Виявлені на етапі налагодження проблеми зі швидкодією було усунено, після чого робота пристрою стала стабільною.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі описано повний шлях розробки апаратного контролера керування мультимедіа – від формування вимог до перевірки готового макета в роботі.

Спочатку на основі висновків першого розділу сформовано перелік функціональних вимог: вісім поворотних енкодерів, дванадцять програмованих кнопок, робота через USB одночасно як HID-пристрій і як джерело подій для драйвера, інформаційний дисплей, живлення одним кабелем та бюджет у межах 30 доларів. Окремою наскрізною вимогою стала повторюваність – здатність зібрати пристрій вручну із загальнодоступних компонентів.

Розроблено структурну та принципову електричну схеми пристрою. Ключовим інженерним рішенням стало зчитування 24 ліній восьми енкодерів через три зсувні регістри 74HC165, з'єднані паралельно, що дозволяє робити одночасний «знімок» усіх фаз – без цього надійне декодування квадратурного сигналу при швидкому обертанні було б неможливим. Обґрунтовано вибір елементної бази: плати Arduino Pro Micro на ATmega32U4 з апаратним USB HID, енкодерів типу EC11, перемикачів Cherry MX Red з діодами 1N4148W

проти фантомних натискань, дисплея 20×4 з інтерфейсом I2C. Сумарну вартість компонентів утримано на рівні близько 32 доларів.

За схемою розведено двошарову друковану плату в KiCad і замовлено її виготовлення, а в SolidWorks спроєктовано та надруковано з PLA роз'ємний корпус із двох деталей. Пристрій зібрано поетапно й перевірено в роботі.

Написано програмне забезпечення з двох частин – мікропрограму для мікроконтролера й програму-драйвер для комп'ютера – з трьома режимами роботи. Проведені експерименти підтвердили працездатність макета: усунено проблеми зі швидкодією, досягнуто надійного зчитування енкодерів, керування гучністю окремих застосунків та коректної роботи в усіх трьох режимах. Розроблений пристрій повністю відповідає сформованим вимогам.

Мікропрограма і програма-драйвер опубліковані разом з вихідними кодами, схема та модель корпусу постачаються у форматах, які приймають вільні засоби проєктування (KiCad для електроніки і будь-який сучасний CAD-пакет для STL-файлу корпусу). Завдяки цьому проєкт може бути повторений, модифікований або портований на іншу платформу без юридичних чи технічних бар'єрів. Така відкритість прямо відповідає на проблему, виявлену в першому розділі на прикладі GoXLR, коли припинення підтримки виробником зробило пристрій непридатним до використання.

Запас для розширення в пристрої закладено: вільні виводи мікроконтролера дозволяють додати додаткові датчики чи індикатори, незалежна пам'ять EEPROM може зберігати більше профілів конфігурації, а модульна архітектура драйвера спрощує додавання нових типів дій або підтримку інших операційних систем. Повноцінний апаратно-програмний контролер мультимедіа реалізовано на загальнодоступних компонентах вартістю близько 32 доларів, з повністю відкритим вихідним кодом і можливістю самостійного повторення.

ВИСНОВКИ

У дипломному проєкті розроблено, виготовлено та протестовано апаратний контролер для керування мультимедіа персонального комп'ютера. Метою роботи було створення компактного настільного пристрою, який працює через єдиний кабель USB одночасно як джерело подій для драйвера та як стандартний HID-пристрій, не потребуючи встановлення додаткового програмного забезпечення.

У першому розділі проведено огляд існуючих рішень у сфері апаратного керування мультимедіа персонального комп'ютера. Детально розглянуто три ключові пристрої — Elgato Stream Deck, TC-Helicon GoXLR та відкритий проєкт deej — їхні сильні сторони та обмеження. На основі порівняльного аналізу сформовано перелік функціональних вимог до власного пристрою: не менше восьми поворотних регуляторів, не менше дванадцяти програмованих кнопок, прямий доступ до системних аудіосесій замість фіксованих віртуальних каналів, повністю відкритий код мікропрограми та клієнта, орієнтовна сумарна вартість компонентів у межах 30 доларів США.

Друга частина роботи зосереджена на практичній реалізації. Обґрунтовано вибір елементної бази: плати Arduino Pro Micro на мікроконтролері ATmega32U4 з апаратною підтримкою USB HID, восьми поворотних енкодерів типу EC11 з вбудованими кнопками, трьох зсувних регістрів 74HC165 для розширення кількості входів, дванадцяти механічних перемикачів Cherry MX Red з діодами 1N4148W для запобігання фантомним натисканням, символьного дисплея 20×4 з інтерфейсом I2C. Розроблено принципову електричну схему й виконано розведення двошарової друкованої плати у середовищі KiCad. У системі SolidWorks спроєктовано та надруковано з пластику PLA роз'ємний корпус із двох деталей зі з'єднанням защіпками.

Програмне забезпечення поділено на дві частини. На стороні мікроконтролера мікропрограму написано мовою C++ у середовищі PlatformIO: вона опитує енкодери через зсувні регістри, сканує матрицю кнопок і паралельно обмінюється даними з персональним комп'ютером через віртуальний послідовний порт. У двох автономних режимах мікропрограма формує події HID самостійно. Супутню програму-драйвер написано мовою Python з графічним інтерфейсом на бібліотеці Kivy. Вона регулює гучність окремих застосунків через інтерфейс Windows Audio Session API, виконує макроси, запускає програми та виводить на дисплей пристрою поточне навантаження системи.

Перевірка працездатності підтвердила, що макетний зразок виконує всі закладені функції. Усунено проблему пропуску кроків при швидкому обертанні енкодерів — частота опитування зросла з 96 до кількох тисяч разів на секунду. У режимі через драйвер пристрій плавно регулює гучність окремих застосунків кроком 1 %. У режимах HID-клавіатури та ігрового контролера кнопки й повороти відображаються на стандартні події системи без додаткового програмного забезпечення. Стабільна робота підтверджена восьмигодинним безперервним тестом при споживанні близько 60–80 мА від шини USB.

Сумарна вартість компонентів становить близько 32 доларів США, що на порядок нижче за найдоступніші комерційні аналоги. Мікропрограму, програму-драйвер, файли плати та моделі корпусу опубліковано з відкритим вихідним кодом, що дозволяє повторити або доопрацювати пристрій без юридичних чи технічних обмежень. На відміну від комерційних аналогів на кшталт GoXLR, які після припинення підтримки виробником стають непрацездатними, цей пристрій залишається повністю керованим власником.

Зібраний прототип демонструє, що повноцінний контролер мультимедіа з кількома режимами роботи реалізується на простих, загальнодоступних

компонентах. Запас за виводами мікроконтролера та модульна архітектура драйвера залишають місце для подальшого розширення — нових типів дій, портування драйвера на Linux і macOS, інтеграції з професійним програмним забезпеченням для звуку та відео.

ПЕРЕЛІК ПОСИЛАНЬ

1. Adjust the volume on Windows : офіційна довідка Microsoft. URL: https://support.microsoft.com/en-us/windows (дата звернення: 05.03.2026).
2. Mathias K. Audio mixing console basics : how to operate a mixer for live sound and studio recording : навчальна стаття Audio University. URL: https://audiouniversityonline.com/mixing-console-tutorial/ (дата звернення: 09.03.2026).
3. Harel O. deej - an open-source hardware volume mixer for Windows and Linux : репозиторій GitHub. URL: https://github.com/omriharel/deej (дата звернення: 17.04.2026).
4. Rotary encoder in CircuitPython : навчальний матеріал Adafruit Learning System. URL: https://learn.adafruit.com/rotary-encoder/overview (дата звернення: 14.04.2026).
5. Multi Actions - Stream Deck documentation : офіційна довідка Elgato. URL: https://help.elgato.com/hc/en-us/articles/360027960912-Elgato-Stream-Deck-Multi-Actions (дата звернення: 22.03.2026).
6. Stream Deck - Product Family Overview : офіційний сайт Elgato. URL: https://www.elgato.com/en/stream-deck (дата звернення: 12.03.2026).
7. Elgato Stream Deck Classic Teardown : iFixit. URL: https://www.ifixit.com/Teardown/Elgato+Stream+Deck+Classic+Teardown/194063 (дата звернення: 18.03.2026).
8. GoXLR : офіційна сторінка продукту TC Helicon. URL: https://www.tchelicon.com/en/products/0803-AAA (дата звернення: 28.03.2026).
9. The GoXLR is dead, but possibly open source : Stream Guides. URL: https://streamguides.gg/2023/07/the-goxlr-is-dead-but-possibly-open-source/ (дата звернення: 03.04.2026).

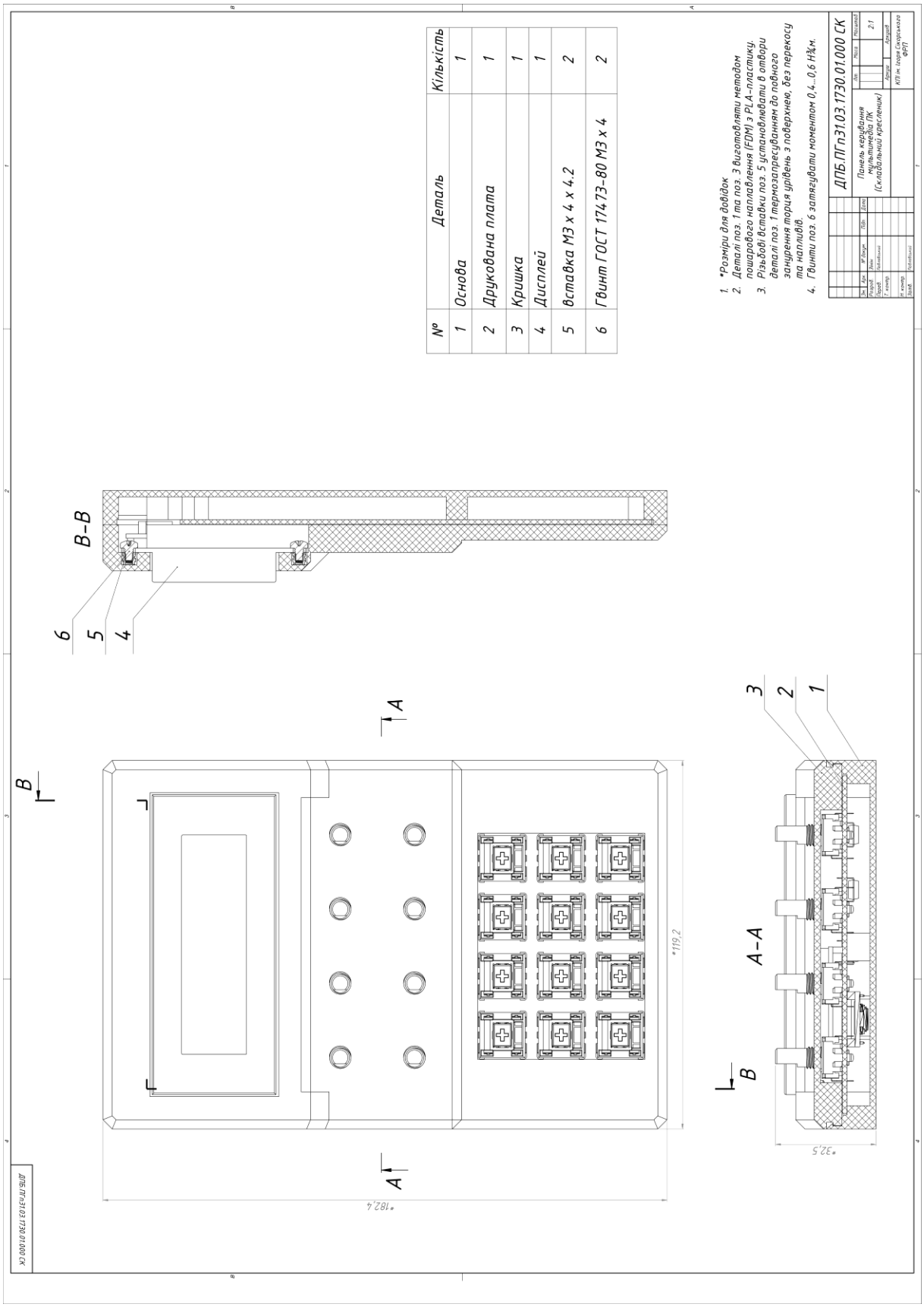
10. goxlr-utility - an unofficial GoXLR utility : репозиторій GitHub. URL: https://github.com/GoXLR-on-Linux/goxlr-utility (дата звернення: 09.04.2026).
11. ATmega16U4/32U4 : 8-bit AVR Microcontroller with USB : datasheet : Microchip Technology. URL: https://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf (дата звернення: 02.05.2026).
12. Pro Micro & Fio V3 Hookup Guide : офіційний посібник SparkFun Electronics. URL: https://learn.sparkfun.com/tutorials/pro-micro--fio-v3-hookup-guide/all (дата звернення: 02.05.2026).
13. Keyboard : USB HID : офіційна довідка Arduino. URL: https://docs.arduino.cc/language-reference/en/functions/usb/Keyboard/ (дата звернення: 04.05.2026).
14. SNx4HC165 8-Bit Parallel-Load Shift Registers : datasheet : Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/sn74hc165.pdf (дата звернення: 05.05.2026).
15. Incremental encoder : Wikipedia. URL: https://en.wikipedia.org/wiki/Incremental_encoder (дата звернення: 06.05.2026).
16. PEC11R Series : 12 mm Incremental Encoder : datasheet : Bourns Inc. URL: https://www.bourns.com/docs/Product-Datasheets/PEC11R.pdf (дата звернення: 06.05.2026).
17. Surface-mount technology : Wikipedia. URL: https://en.wikipedia.org/wiki/Surface-mount_technology (дата звернення: 07.05.2026).
18. CHERRY MX RED : технічний опис механічного перемикача : офіційний сайт Cherry. URL: https://www.cherry.de/en-us/product/mx2a-red (дата звернення: 08.05.2026).

19. Dribin D. Keyboard Matrix Help : про сканування матриці та фантомні натискання. URL: https://www.dribin.org/dave/keyboard/one_html/ (дата звернення: 08.05.2026).
20. 1N4148W : Surface Mount Fast Switching Diode (SOD-123) : datasheet : Diodes Incorporated. URL: https://www.diodes.com/assets/Datasheets/BAV16W_1N4148W.pdf (дата звернення: 09.05.2026).
21. UM10204 : I2C-bus specification and user manual : NXP Semiconductors. URL: https://www.nxp.com/docs/en/user-guide/UM10204.pdf (дата звернення: 10.05.2026).
22. I2C Serial Interface 2004 LCD Module : специфікація модуля : Handson Technology. URL: https://www.handsontec.com/dataspecs/I2C_2004_LCD.pdf (дата звернення: 10.05.2026).
23. KiCad EDA : A Cross Platform and Open Source Electronics Design Automation Suite : офіційний сайт. URL: https://www.kicad.org/ (дата звернення: 11.05.2026).
24. SOLIDWORKS : 3D CAD Design Software : офіційний сайт Dassault Systèmes. URL: https://www.solidworks.com/ (дата звернення: 11.05.2026).
25. Fused filament fabrication : Wikipedia. URL: https://en.wikipedia.org/wiki/Fused_filament_fabrication (дата звернення: 12.05.2026).
26. PlatformIO : a professional collaborative platform for embedded development : офіційний сайт. URL: https://platformio.org/ (дата звернення: 12.05.2026).
27. NicoHood. HID-Project : бібліотека USB HID для Arduino : репозиторій GitHub. URL: https://github.com/NicoHood/HID (дата звернення: 14.05.2026).

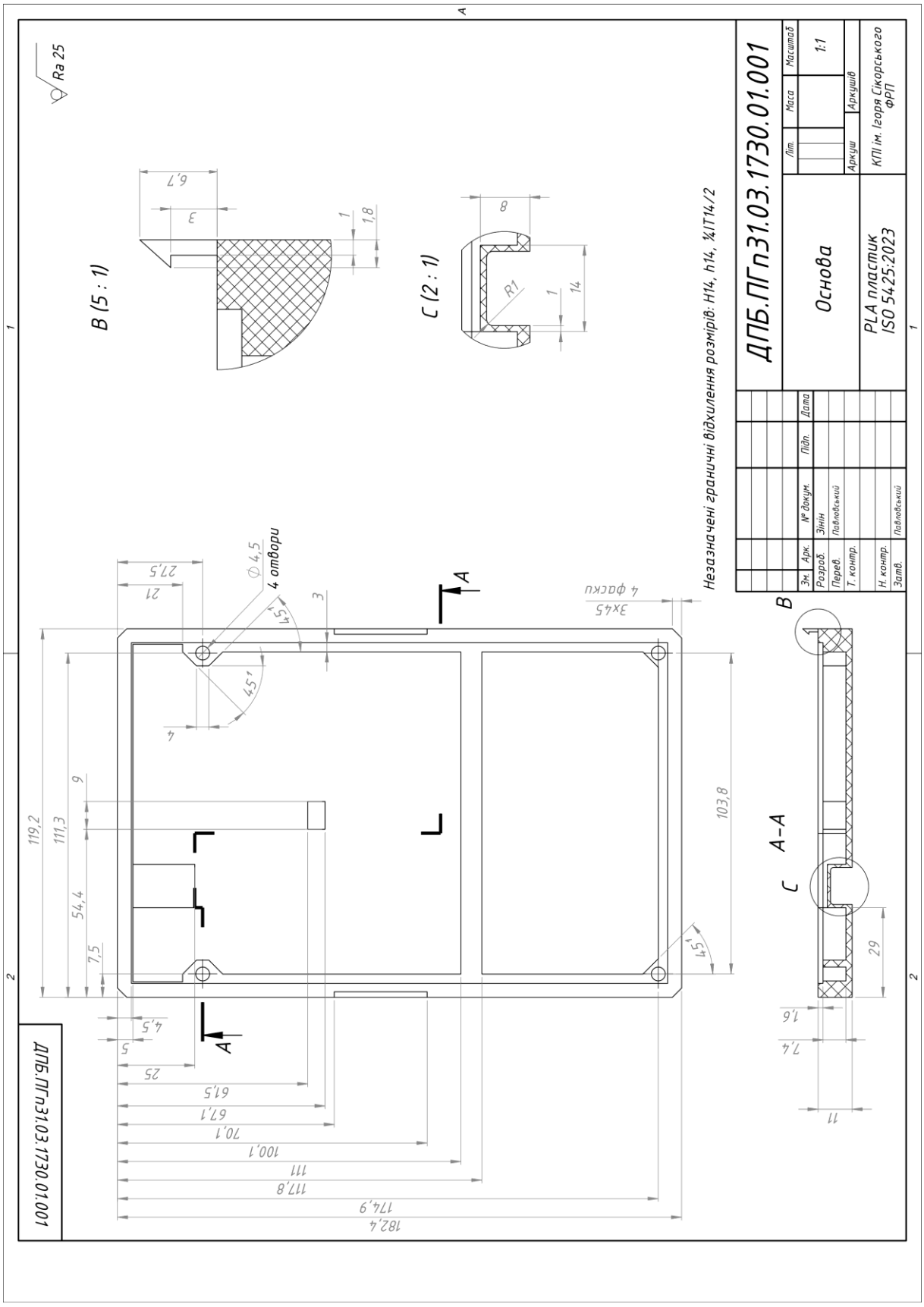
28. luni64. EncoderTool : бібліотека опрацювання поворотних енкодерів : репозиторій GitHub. URL: https://github.com/luni64/EncoderTool (дата звернення: 14.05.2026).
29. Fredericks T. Bounce2 : бібліотека усунення деренчання контактів : репозиторій GitHub. URL: https://github.com/thomasfredericks/Bounce2 (дата звернення: 15.05.2026).
30. Schwartz M., de Brabander F. LiquidCrystal_I2C : бібліотека керування I2C-дисплеями : репозиторій GitHub. URL: https://github.com/marcoschwartz/LiquidCrystal_I2C (дата звернення: 13.05.2026).
31. Language Reference : офіційна документація Arduino. URL: https://docs.arduino.cc/language-reference/ (дата звернення: 13.05.2026).
32. WASAPI : Windows Audio Session API : офіційна документація Microsoft. URL: https://learn.microsoft.com/en-us/windows/win32/coreaudio/wasapi (дата звернення: 15.05.2026).
33. Miras A. pycaw : Python Core Audio Windows Library : репозиторій GitHub. URL: https://github.com/AndreMiras/pycaw (дата звернення: 16.05.2026).
34. Rodola G. psutil : Cross-platform process and system utilities for Python : репозиторій GitHub. URL: https://github.com/giampaolo/psutil (дата звернення: 16.05.2026).
35. Kivy : Open source Python app development framework : офіційний сайт. URL: https://kivy.org/ (дата звернення: 17.05.2026).
36. pySerial : документація бібліотеки. URL: https://pyserial.readthedocs.io/en/latest/ (дата звернення: 17.05.2026).
37. boppreh. keyboard : Hook and simulate keyboard events on Windows and Linux : репозиторій GitHub. URL: https://github.com/boppreh/keyboard (дата звернення: 18.05.2026).

38. Performance Counters : Win32 apps : офіційна документація Microsoft. URL: https://learn.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal (дата звернення: 18.05.2026).
39. Hot air solder leveling : Wikipedia. URL: https://en.wikipedia.org/wiki/Hot_air_solder_leveling (дата звернення: 09.06.2026).

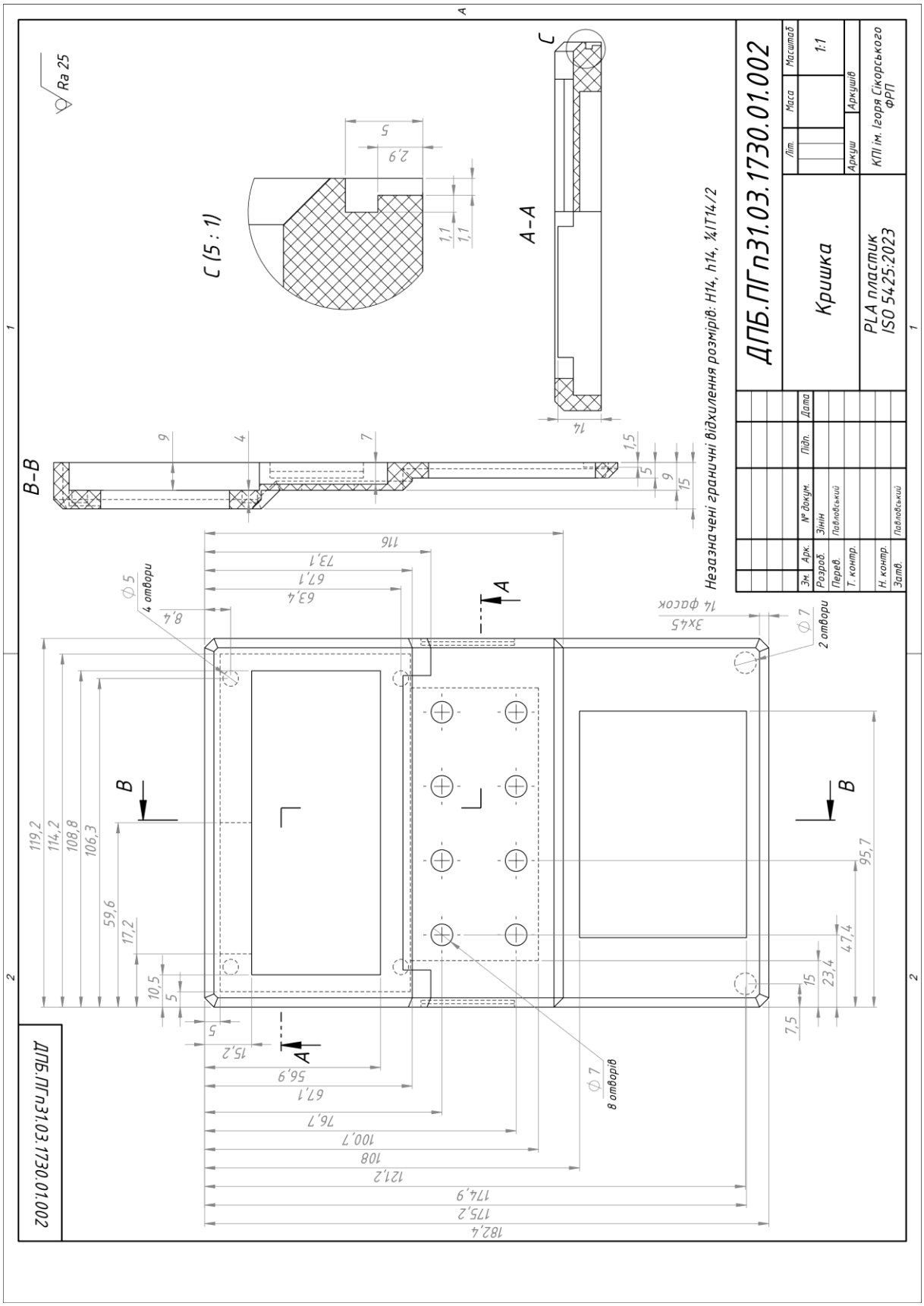
ДОДАТОК А


ДПБ.ПГ.п31.03.1730.01.000 СК
Панель керування мультиметра ПК (Складальний креслення)
КПІ ім. Ігоря Сікорського ФІОТ
1. *Розміри для довідок
2. Деталі поз. 1 та поз. 3 виготовляти методом пошарового наплавлення (FDM) з PLA-пластику.
3. Різьбові вставки поз. 5 установлювати в отвори деталі поз. 1 термозатресуванням до повного занурення торця уровень з поверхнею, без перекосу та напливів.
4. Гвинти поз. 6 затягувати моментом 0,4...0,6 Н*м.
№ | Деталь | Кількість
1 | Основа | 1
2 | Друкована плата | 1
3 | Кришка | 1
4 | Дисплей | 1
5 | Вставка М3 х 4 х 4.2 | 2
6 | Гвинт ГОСТ 17473-80 М3 х 4 | 2
B–B
A–A
*182,4
*119,2
*32,5

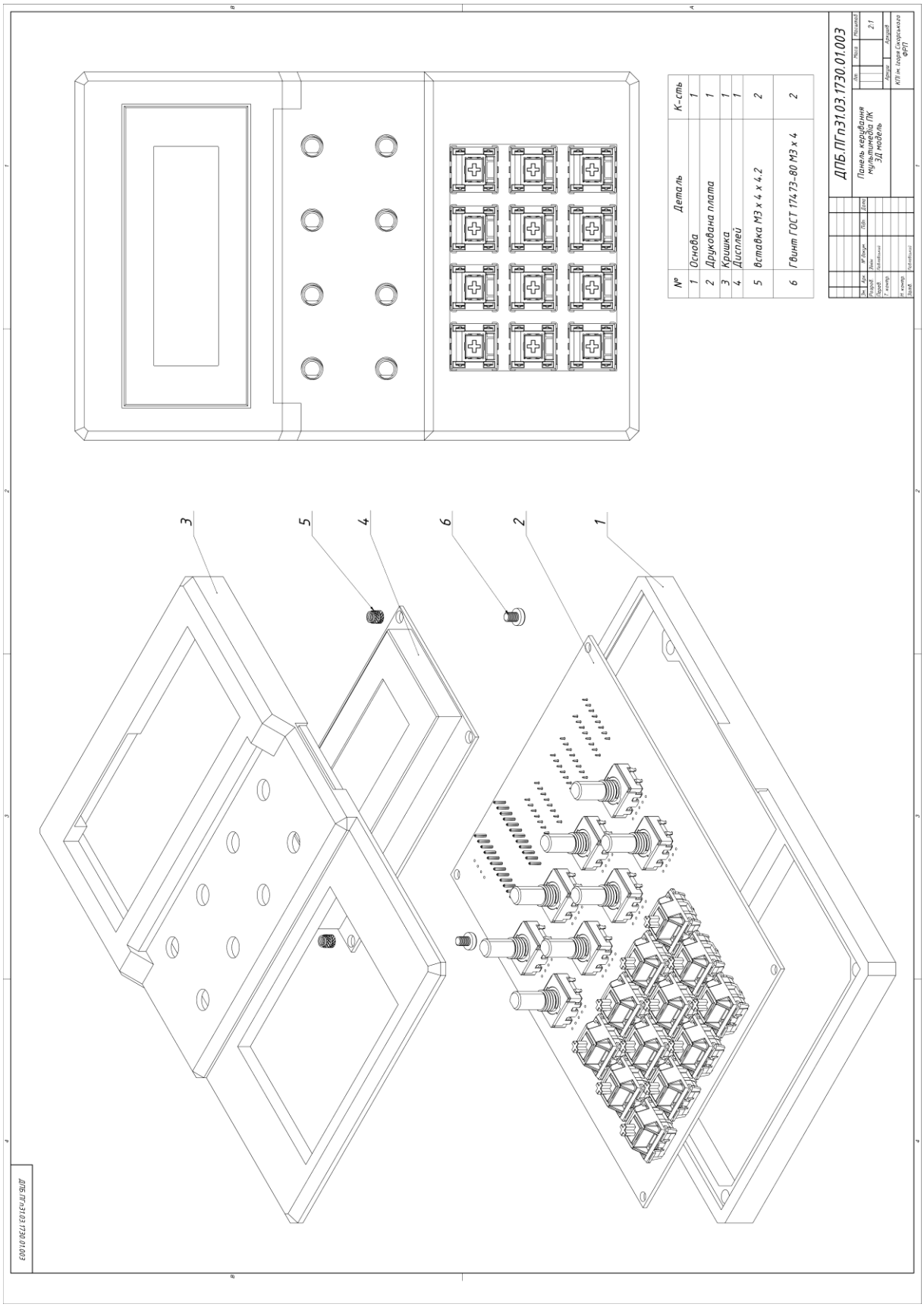
ДОДАТОК Б


Ra 25
B (5 : 1)
C (2 : 1)
A–A
Незазначені граничні відхилення розмірів: H14, h14, ±IT14/2
4 отвори
4 фаски
3x45
ДПБ.ПГп31.03.1730.01.001
Основа
PLA пластик
ISO 5425:2023
КПІ ім. Ігоря Сікорського ФРП
Масштаб
1:1
Маса
Лит.
Аркуш
Аркушів
Зм. Арк. № докум. Підп. Дата
Розроб. Зінін
Перев. Павловський
Т. контр.
Н. контр.
Затв. Павловський
ДПБ.ПГп31.03.1730.01.001

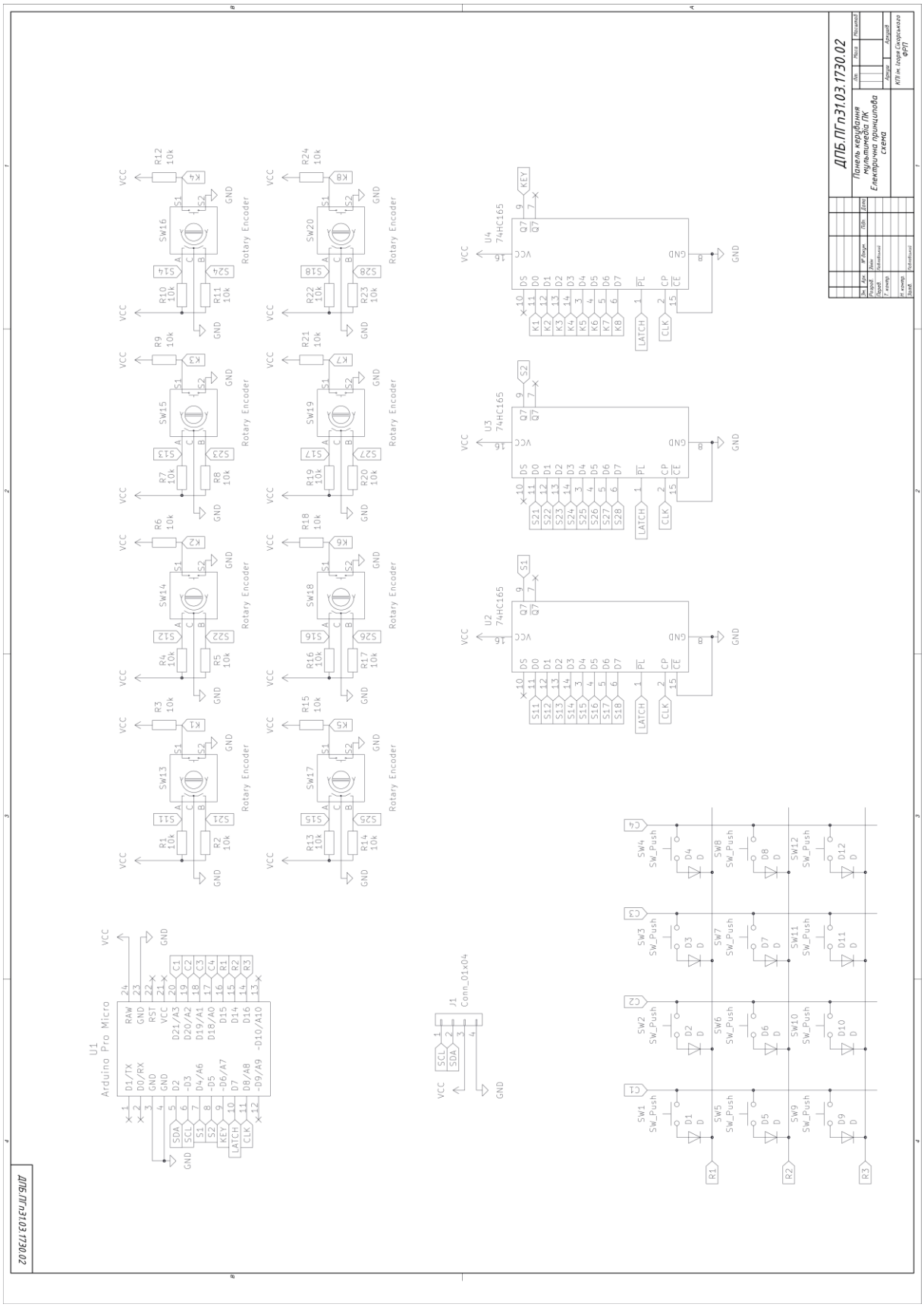
ДОДАТОК В


Ra 25
С (5 : 1)
А–А
В–В
Незазначені граничні відхилення розмірів: H14, h14, ±IT14/2
ДПБ.ПГп31.03.1730.01.002
Кришка
PLA пластик
ISO 5425:2023
КПІ ім. Ігоря Сікорського ФРП
Розроб. Зінін
Перев. Павловський
Т. контр.
Н. контр. Павловський
Затв.
Масштаб 1:1
Маса
Аркуш
Аркушів
Літ.
№ докум.
Підп.
Дата
Зм. Арк.
3х45
14 фасок
4 отвори
2 отвори
8 отворів
ДПБ.ПГп31.03.1730.01.002

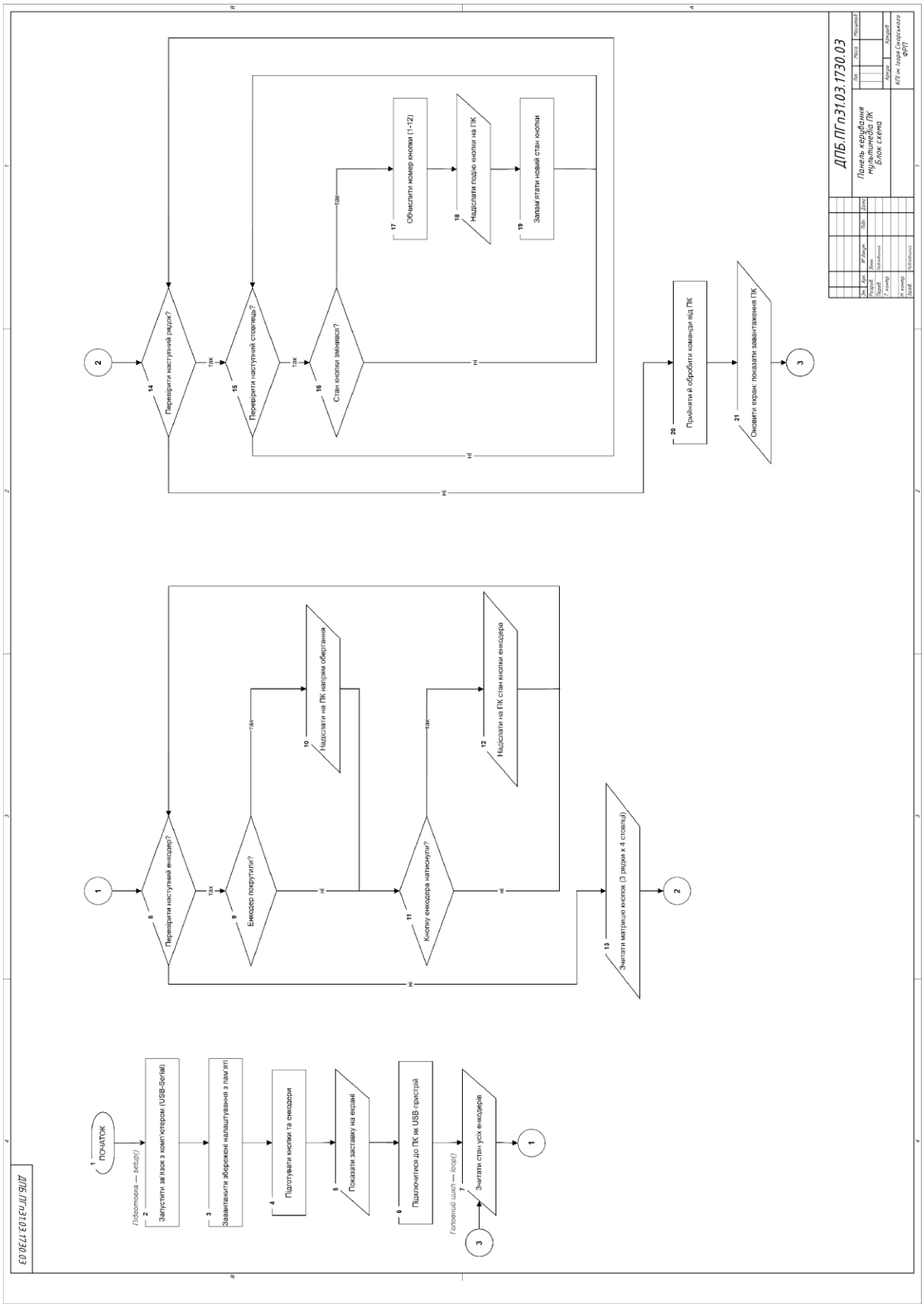
ДОДАТОК Г


№ | Деталь | К-сть
1 | Основа | 1
2 | Друкована плата | 1
3 | Кришка | 1
4 | Дисплей | 1
5 | Вставка М3 х 4 х 4.2 | 2
6 | Гвинт ГОСТ 17473-80 М3 х 4 | 2
ДПБ.ПГп31.03.1730.01.003
Панель керування мультиметра ПК 3Д модель
КПІ ім. Ігоря Сікорського ФІОТ
3
5
4
6
2
1

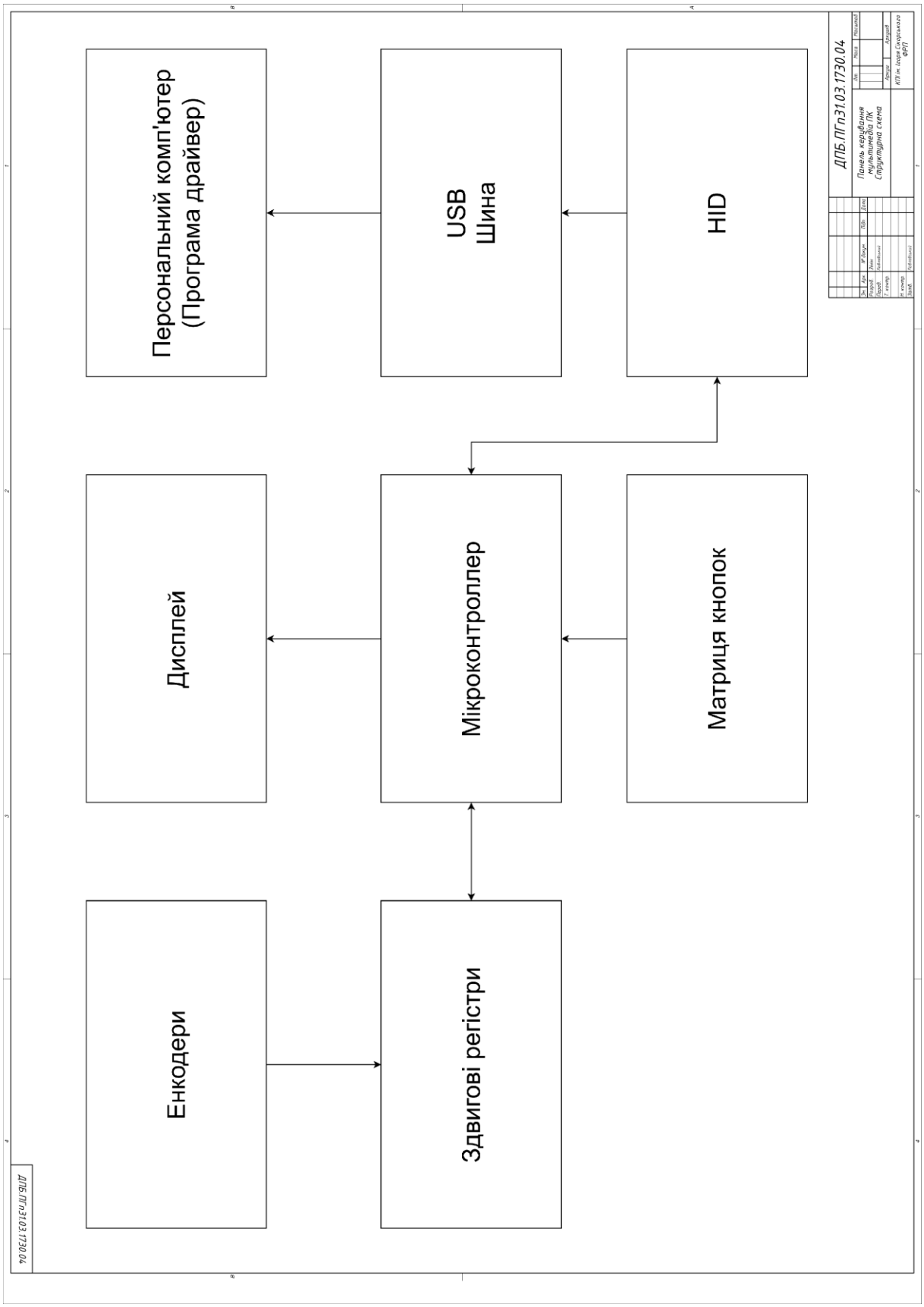
ДОДАТОК Ґ


ДПБ.ПЛ.п31.03.1730.02
Панель керування мультиметра ПК
Електричні принципові схеми
КПІ ім. Ігоря Сікорського ФІОТ
Arduino Pro Micro
74HC165
Rotary Encoder
SW_Push

ДОДАТОК Д


ДПБ.ПГп31.03.1730.03
Панель керування мультиметра ПК
Блок схема
КПІ ім. Ігоря Сікорського ФІОТ
ПОЧАТОК
Підготовка — setup()
Головний цикл — loop()
Запустити зв'язок з комп'ютером (USB-Serial)
Завантажити збережені налаштування з пам'яті
Підготувати кнопки та енкодери
Показати заставку на екрані
Підключитися до ПК як USB-пристрій
Зчитати стан усіх енкодерів
Перевірити наступний енкодер?
Енкодер покрутили?
Надіслати на ПК напрям обертання
Кнопку енкодера натиснули?
Надіслати на ПК стан кнопки енкодера
Зчитати матрицю кнопок (3 рядки × 4 стовпці)
Перевірити наступний рядок?
Перевірити наступний стовпець?
Стан кнопки змінився?
Обчислити номер кнопки (1–12)
Надіслати подію кнопки на ПК
Запам'ятати новий стан кнопки
Прийняти й обробити команди від ПК
Оновити екран: показати завантаження ПК
так
ні

ДОДАТОК Е


Персональний комп'ютер
(Програма драйвер)
USB
Шина
HID
Дисплей
Мікроконтроллер
Матриця кнопок
Енкодери
Здвигові регістри
ДПБ.ПГп31.03.1730.04
ДПБ.ПГп31.03.1730.04
Панель керування
мультимедіа ПК
Структурна схема

ДОДАТОК Є

```
#include "Arduino.h"

#include "HID-Project.h"

#include <Wire.h>

#include <LiquidCrystal_I2C.h>


#include "hardware_conf.h"

#include "EEPROMConfig.h"

#include "Hardware.h"



LiquidCrystal_I2C lcd(0x27, 20, 4);



void checkModeSwitch();

void processKeyEvents();

void processEncoderEvents();

void handleSerialCommands();
```

```
void updateLCD();



void onHIDPress(uint8_t actionIdx);

void onHIDRelease(uint8_t actionIdx);

void onHIDPulse(uint8_t actionIdx);



inline void sPrint(const char* s)        { if (Serial) Serial.print(s); }

inline void sPrint(uint8_t v)            { if (Serial) Serial.print(v); }

inline void sPrintln(const char* s = "") { if (Serial) Serial.println(s); }

inline void sPrintln(uint8_t v)          { if (Serial) Serial.println(v); }



bool          modeSwitchLatch = false;

unsigned long lastModeSwitchMs = 0;



uint8_t       gpPulseBtn = 0;

unsigned long gpPulseMs  = 0;
```

```
unsigned long lcdLastMs = 0;
uint8_t       lcdPrevMode = 0;
char          lcdLine0[21] = "";
char          lcdLine1[21] = "";
char          lcdLine2[21] = "";
char          lcdLine3[21] = "";


struct SystemStats {
    uint8_t  cpu;
    uint8_t  ram;
    uint8_t  gpu;
    uint8_t  dsk;
    uint16_t netRx;
    uint16_t netTx;
    unsigned long lastUpdateMs;
};
SystemStats sysStats = {};
const unsigned long SYS_STALE_MS = 3000;
```

```
//
================================================================================
================================

void setup()
{
    Serial.begin(115200);
    delay(200);

    loadConfig();
    initHardware();

    lcd.init();
    lcd.backlight();
    lcd.setCursor(2, 1);
    lcd.print("Media Panel V2.0");
    lcd.setCursor(2, 2);
    lcd.print("Zinin OA  PG-p31");
    delay(2000);
    lcd.clear();

    Keyboard.begin();
```

```
    Consumer.begin();
    Gamepad.begin();

    sPrint("E:M:"); sPrintln(currentMode);
}

void loop()
{
    encoders.tick();
    processEncoderEvents();

    if (gpPulseBtn > 0 && millis() - gpPulseMs >= 30) {
        Gamepad.release(gpPulseBtn);
        Gamepad.write();
        gpPulseBtn = 0;
    }

    static unsigned long timer = 0;
    if (millis() - timer >= 10) {
        timer = millis();
```

```
        scanKeyMatrix();
        checkModeSwitch();
        processKeyEvents();
    }

    handleSerialCommands();



    if (millis() - lcdLastMs >= 250) {
        lcdLastMs = millis();
        updateLCD();
    }
}

void checkModeSwitch()
{
    bool b11 = keyState[2][2];
    bool b12 = keyState[2][3];

    if (b11 && b12) {
```

```
        if (!modeSwitchLatch && millis() - lastModeSwitchMs > 500) {
            modeSwitchLatch  = true;
            lastModeSwitchMs = millis();
            currentMode = (currentMode % 3) + 1;
            sPrint("E:M:"); sPrintln(currentMode);
        }
    } else {
        modeSwitchLatch = false;
    }
}

void processKeyEvents()
{
    for (uint8_t r = 0; r < ROWS; r++) {
        for (uint8_t c = 0; c < COLS; c++) {
            if (keyState[r][c] == keyPrev[r][c]) continue;

            uint8_t keyNum   = r * COLS + c + 1;
            uint8_t actionIdx = keyNum - 1;
            bool pressed = keyState[r][c];
```

```
            if ((keyNum == 11 || keyNum == 12) && modeSwitchLatch) {
                keyPrev[r][c] = keyState[r][c];
                continue;
            }

            sPrint("E:B:"); sPrint(keyNum);
            sPrint(":"); sPrintln(pressed ? (uint8_t)1 : (uint8_t)0);

            if (currentMode >= 2) {
                if (pressed) onHIDPress(actionIdx);
                else         onHIDRelease(actionIdx);
            }

            keyPrev[r][c] = keyState[r][c];
        }
    }
}

void processEncoderEvents()
```

```
{
    for (unsigned i = 0; i < encoderCount; i++) {
        unsigned phys = physicalEnc(i) + 1;

        if (encoders[i].valueChanged()) {
            bool right = encoders[i].isRight();
            sPrint("E:R:"); sPrint((uint8_t)phys);
            sPrint(":"); sPrintln(right ? "R" : "L");

            if (currentMode >= 2) {
                uint8_t idx = right ? (12 + phys - 1) : (20 + phys - 1);
                onHIDPulse(idx);
            }
        }

        if (encoders[i].buttonChanged()) {
            bool pressed = (encoders[i].getButton() == 0);
            sPrint("E:P:"); sPrint((uint8_t)phys);
            sPrint(":"); sPrintln(pressed ? (uint8_t)1 : (uint8_t)0);
```

```
            if (currentMode >= 2) {
                uint8_t idx = 28 + phys - 1;
                if (pressed) onHIDPress(idx);
                else         onHIDRelease(idx);
            }
        }
    }
}

static void applyModifiers(uint8_t mod)
{
    if (mod & MOD_CTRL)  Keyboard.press(KEY_LEFT_CTRL);
    if (mod & MOD_SHIFT) Keyboard.press(KEY_LEFT_SHIFT);
    if (mod & MOD_ALT)   Keyboard.press(KEY_LEFT_ALT);
    if (mod & MOD_GUI)   Keyboard.press(KEY_LEFT_GUI);
}

void onHIDPress(uint8_t idx)
{
    if (idx >= TOTAL_ACTIONS) return;
```

```
    if (currentMode == 2) {
        HIDAction& a = mode2Config[idx];
        if (a.type == ACT_KEYBOARD) {
            applyModifiers(a.mod);
            Keyboard.press(KeyboardKeycode(a.code));
        } else if (a.type == ACT_CONSUMER) {
            uint16_t cc = ((uint16_t)a.mod << 8) | a.code;
            Consumer.press(ConsumerKeycode(cc));
        }
    }
    else if (currentMode == 3) {
        uint8_t btn = idx + 1;
        if (btn <= 32) {
            Gamepad.press(btn);
            Gamepad.write();
        }
    }
}
```

```
void onHIDRelease(uint8_t idx)
{
    if (idx >= TOTAL_ACTIONS) return;

    if (currentMode == 2) {
        Keyboard.releaseAll();
        Consumer.releaseAll();
    }
    else if (currentMode == 3) {
        uint8_t btn = idx + 1;
        if (btn <= 32) {
            Gamepad.release(btn);
            Gamepad.write();
        }
    }
}

void onHIDPulse(uint8_t idx)
{
    if (idx >= TOTAL_ACTIONS) return;
```

```
if (currentMode == 2) {
    HIDAction& a = mode2Config[idx];
    if (a.type == ACT_KEYBOARD) {
        applyModifiers(a.mod);
        Keyboard.press(KeyboardKeycode(a.code));
        delay(10);
        Keyboard.releaseAll();
    } else if (a.type == ACT_CONSUMER) {
        uint16_t cc = ((uint16_t)a.mod << 8) | a.code;
        Consumer.write(ConsumerKeycode(cc));
    }
}
else if (currentMode == 3) {
    uint8_t btn = idx + 1;
    if (btn <= 32) {
        if (gpPulseBtn > 0 && gpPulseBtn != btn) {
            Gamepad.release(gpPulseBtn);
        }
        Gamepad.press(btn);
```

```
            Gamepad.write();
            gpPulseBtn = btn;
            gpPulseMs  = millis();
        }
    }
}

static char cmdBuf[64];
static uint8_t cmdLen = 0;

void handleSerialCommands()
{
    while (Serial.available()) {
        char c = Serial.read();
        if (c == '\n' || c == '\r') {
            if (cmdLen == 0) continue;
            cmdBuf[cmdLen] = '\0';
            cmdLen = 0;
            break;
        }
```

```
    if (cmdLen < sizeof(cmdBuf) - 1) {
        cmdBuf[cmdLen++] = c;
    }
    return;
}
if (cmdBuf[0] == '\0') return;

String cmd(cmdBuf);
cmdBuf[0] = '\0';

if (cmd == "GM") {
    Serial.print("CM:"); Serial.println(currentMode);
    return;
}

if (cmd.startsWith("SM:")) {
    uint8_t m = cmd.substring(3).toInt();
    if (m >= 1 && m <= 3) {
        currentMode = m;
        Serial.print("E:M:"); Serial.println(currentMode);
```

```
    }
    return;
}

if (cmd == "GA") {
    for (uint8_t i = 0; i < TOTAL_ACTIONS; i++) {
        Serial.print("CD:"); Serial.print(i);
        Serial.print(":2:");
        Serial.print(mode2Config[i].type); Serial.print(":");
        Serial.print(mode2Config[i].code); Serial.print(":");
        Serial.println(mode2Config[i].mod);
        Serial.print("CD:"); Serial.print(i);
        Serial.print(":3:");
        Serial.print(mode3Config[i].type); Serial.print(":");
        Serial.println(mode3Config[i].button);
    }
    Serial.print("CM:"); Serial.println(currentMode);
    Serial.println("CE");
    return;
}
```

```
if (cmd.startsWith("CS:")) {
    int p1 = 3;
    int p2 = cmd.indexOf(':', p1);
    int p3 = cmd.indexOf(':', p2 + 1);
    int p4 = cmd.indexOf(':', p3 + 1);
    int p5 = cmd.indexOf(':', p4 + 1);
    if (p2 < 0 || p3 < 0 || p4 < 0) return;

    uint8_t idx  = cmd.substring(p1, p2).toInt();
    uint8_t mode = cmd.substring(p2 + 1, p3).toInt();
    uint8_t type = cmd.substring(p3 + 1, p4).toInt();
    uint8_t v1   = cmd.substring(p4 + 1, (p5 > 0 ? p5 : cmd.length())).toInt();
    uint8_t v2   = (p5 > 0) ? cmd.substring(p5 + 1).toInt() : 0;

    setActionConfig(idx, mode, type, v1, v2);
    Serial.println("OK");
    return;
}
```

```
if (cmd == "CW") {
    saveConfig();
    Serial.println("OK");
    return;
}

if (cmd.startsWith("SL:")) {
    int p1 = 3;
    int p2 = cmd.indexOf(':', p1);
    int p3 = cmd.indexOf(':', p2 + 1);
    int p4 = cmd.indexOf(':', p3 + 1);
    int p5 = cmd.indexOf(':', p4 + 1);
    int p6 = cmd.indexOf(':', p5 + 1);
    if (p2 > 0 && p3 > 0 && p4 > 0 && p5 > 0 && p6 > 0) {
        uint16_t cpu = cmd.substring(p1, p2).toInt();
        uint16_t ram = cmd.substring(p2 + 1, p3).toInt();
        uint16_t gpu = cmd.substring(p3 + 1, p4).toInt();
        uint16_t dsk = cmd.substring(p4 + 1, p5).toInt();
        uint16_t rx  = cmd.substring(p5 + 1, p6).toInt();
        uint16_t tx  = cmd.substring(p6 + 1).toInt();
```

```
            sysStats.cpu = cpu > 100 ? 100 : cpu;
            sysStats.ram = ram > 100 ? 100 : ram;
            sysStats.gpu = gpu > 100 ? 100 : gpu;
            sysStats.dsk = dsk > 100 ? 100 : dsk;
            sysStats.netRx = rx > 9999 ? 9999 : rx;
            sysStats.netTx = tx > 9999 ? 9999 : tx;
            sysStats.lastUpdateMs = millis();
        }
        return;
    }
}


static uint8_t lcdNextRow = 0;
static char    lcdPending[4][21];

static void fmtPct3(char* dst, uint8_t v)
{
    if (v > 100) v = 100;
    if (v >= 100) {
```

```
        dst[0] = '1'; dst[1] = '0'; dst[2] = '0';
    } else if (v >= 10) {
        dst[0] = ' '; dst[1] = '0' + v / 10; dst[2] = '0' + v % 10;
    } else {
        dst[0] = ' '; dst[1] = ' '; dst[2] = '0' + v;
    }
}

static void fmtVal4(char* dst, uint16_t v)
{
    if (v > 9999) v = 9999;
    dst[0] = (v >= 1000) ? ('0' + v / 1000) : ' ';
    dst[1] = (v >= 100)  ? ('0' + (v / 100) % 10) : ' ';
    dst[2] = (v >= 10)   ? ('0' + (v / 10) % 10)  : ' ';
    dst[3] = '0' + v % 10;
}

static void lcdPrepareContent()
{
    bool freshData = (millis() - sysStats.lastUpdateMs) < SYS_STALE_MS
```

```
                && sysStats.lastUpdateMs != 0;

    if (currentMode == 1 && freshData) {
        memcpy(lcdPending[0], "CPU:   %  RAM:   %  ", 20);
        fmtPct3(lcdPending[0] + 4,  sysStats.cpu);
        fmtPct3(lcdPending[0] + 14, sysStats.ram);
        lcdPending[0][20] = '\0';

        memcpy(lcdPending[1], "GPU:   %  DSK:   %  ", 20);
        fmtPct3(lcdPending[1] + 4,  sysStats.gpu);
        fmtPct3(lcdPending[1] + 14, sysStats.dsk);
        lcdPending[1][20] = '\0';

        memcpy(lcdPending[2], "DN:     UP:     KB/s", 20);
        fmtVal4(lcdPending[2] + 3,  sysStats.netRx);
        fmtVal4(lcdPending[2] + 11, sysStats.netTx);
        lcdPending[2][20] = '\0';

        memcpy(lcdPending[3], "MacroPanel v2.0  M:1", 20);
        lcdPending[3][19] = '0' + currentMode;
```

```
      lcdPending[3][20] = '\0';
    } else if (currentMode == 1) {
      memcpy(lcdPending[0], "    MacroPanel      ", 20); lcdPending[0][20] = '\0';
      memcpy(lcdPending[1], "  Driver Mode (1)   ", 20); lcdPending[1][20] = '\0';
      memcpy(lcdPending[2], "  waiting for PC... ", 20); lcdPending[2][20] = '\0';
      memcpy(lcdPending[3], "                M:1", 20); lcdPending[3][20] = '\0';
    } else {
      memset(lcdPending[0], ' ', 20); lcdPending[0][20] = '\0';
      memcpy(lcdPending[1], "  Media Panel V2.0  ", 20); lcdPending[1][20] = '\0';
      memcpy(lcdPending[2], "  Zinin OA  PG-p31  ", 20); lcdPending[2][20] = '\0';
      snprintf(lcdPending[3], 21, "               M:%d", currentMode);
    }
}

void updateLCD()
{
    if (lcdPrevMode != currentMode) {
      lcdNextRow = 0;
      lcdPrevMode = currentMode;
    }
```

```
lcdPrepareContent();


uint8_t row = lcdNextRow;
char* cache;
switch (row) {
    case 0: cache = lcdLine0; break;
    case 1: cache = lcdLine1; break;
    case 2: cache = lcdLine2; break;
    default: cache = lcdLine3; break;
}


if (strcmp(cache, lcdPending[row]) != 0) {
    lcd.setCursor(0, row);
    lcd.print(lcdPending[row]);
    uint8_t len = strlen(lcdPending[row]);
    for (uint8_t i = len; i < 20; i++) lcd.print(' ');
    strncpy(cache, lcdPending[row], 20);
    cache[20] = '\0';
}
```

```
    lcdNextRow = (lcdNextRow + 1) & 0x03;

}
```