НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

Виконав: студент 4 курсу, групи ІО-12

Воробйов Антон Русланович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник проф. каф. ОТ, д.т.н., проф. Клименко Ірина Анатоліївна _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас. каф. ОТ, Нікольський Сергій Сергійович _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент проф. кафедри ІСТ, д.т.н., проф. Корнієнко Богдан Ярославович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____ 2025 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Воробйова Антона Руслановича

1. Тема проєкту *Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати*
керівник проєкту Клименко І. А.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23.05 2025 року № 1705с
2. Термін здачі студентом закінченого проєкту 02.06.2025
3. Вихідні дані до проєкту: *технічна документація, теоретичні дані*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд існуючих рішень
Розділ 2. Аналіз предметної області
Розділ 3. Розробка автономної метеостанції на базі STM32

Розділ 4. Розробка навчально-методичного комплексу для проєктування електронних систем

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень)
Принципова схема пристрою, структурна схема пристрою, функціональна схема (діаграма класів), макет друкованої плати.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль			

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	04.04.2025	
2.	*Вивчення та аналіз завдання*	14.04.2025-04.05.2025	
3.	*Розробка архітектури та загальної структури системи*	21.04.2025-01.05.2025	
4.	*Розробка структур окремих підсистем*	02.05.2025-11.05.2025	
5.	*Програмна реалізація системи*	28.04.2025-11.05.2025	
6.	*Оформлення пояснювальної записки*	24.04.2025-01.06.2025	
7.	*Захист програмного продукту*	04.06.2025	
8.	*Передзахист*	09.06.2025	
9.	*Захист*	16.06.2025	

Студент-дипломник _____Антон Воробйов_____
(підпис)

Керівник проєкту _____Ірина Клименко_____
(підпис)

АНОТАЦІЯ

У цій роботі було розроблено навчально-методичний комплекс для проєктування електронних систем, який включає в себе основи створення електричних схем, проектування друкованих плат та програмування мікроконтролерів STM32. Комплекс охоплює усі етапи проєктування та адаптований для самостійного опрацювання студентами.

У рамках проєкту було створено прототип метеостанції. Розроблений прилад збирає дані про температуру, вологу, тиск, освітленість, УФ-випромінювання та вологість поверхні із навколишнього середовища та передає їх на веб-сервер за допомогою стільникового модему. Конструкція пристрою орієнтована на автономну роботу з низьким енергоспоживанням.

Ключові слова: друковані плати, KiCad, STM32, C.

ANNOTATION

This work presents the development of an educational and methodological guide for designing electronic systems, which includes the fundamentals of creating electrical circuits, designing printed circuit boards (PCBs), and programming STM32 microcontrollers. The guide covers all stages of the design process and is adapted for independent student use.

As part of the project, a prototype of a weather station was developed. The device collects data on temperature, humidity, atmospheric pressure, light intensity, ultraviolet radiation, and surface moisture from the environment. The collected information is transmitted to a web server via a cellular modem. The design of the device is optimized for autonomous operation with low power consumption.

Keywords: printed circuit boards, KiCad, STM32, C.

ВІДОМІСТЬ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати*»

Київ – 2025

ОПИС АЛЬБОМУ

з/п	Формат	Позначення	Найменування	Кількість	Примітки
	А4		Документація загальна		
			Знову розроблена		
2	А4	***ІАЛЦ.467200.002 ТЗ***	Навчально-методичний	4	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Технічне завдання		
3	А4	***ІАЛЦ.467200.003 ПЗ***	Навчально-методичний	87	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Пояснювальна записка		
4	А4	***ІАЛЦ.467200.004 ДА***	Навчально-методичний	2	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Принципова схема пристрою		
5	А4	***ІАЛЦ.467200.005 ДБ***	Навчально-методичний	1	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Структурна схема пристрою		
6	А4	***ІАЛЦ.467200.008 ДВ***	Навчально-методичний	1	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Функціональна схема (діаграма		
			класів)		
7	А4	***ІАЛЦ.467200.008 ДГ***	Навчально-методичний	1	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Макет друкованої плати		

Зм.	Арк.	№ докум.	Підп.	Дата	***ІАЛЦ.467200.001 ОА***		
Розроб.		Воробйов А. Р.			*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати* *Опис альбому*	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.					
Реценз.						***НТУУ «КПІ» ФІОТ ІО-12***	
Н. контр.		Нікольський С. С.					
Затверд.							

8	А4	***ІАЛЦ.467200.008 ДҐ***	Навчально-методичний	16	
			комплекс для проєктування		
			електронних систем на базі STM32		
			Текст програмного коду		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати*»

Київ – 2025

ТЕХНІЧНЕ ЗАВДАННЯ

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ 2
3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ 2
4. ДЖЕРЕЛА РОЗРОБКИ 2
5. ТЕХНІЧНІ ВИМОГИ 3
   5.1. Вимоги до розроблюваного продукту 3
   5.2. Вимоги до програмного забезпечення 3
   5.3. Вимоги до апаратної частини 3
6. ЕТАПИ РОЗРОБКИ 4

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання передбачає створення навчального програмно-апаратного комплексу для проєктування електронних систем на базі STM32 із розробкою друкованої плати.

Основним призначенням навчально-методичного комплексу є допомога студентам у засвоєнні принципів проєктування електронних систем, включно з програмуванням мікроконтролеру, розробкою принципових електричних систем, проєктуванням друкованих плат.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки цієї системи є завдання кваліфікаційної роботи рівня «бакалавр» зі спеціальності «Інженерія програмного забезпечення», затверджене факультетом інформатики та обчислювальної техніки кафедри обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою цієї роботи є створення методичного посібника, що описує усі етапи проєктування електронних систем для подальшого використання в освітньому процесі.

4. ДЖЕРЕЛА РОЗРОБКИ

Інформаційною основою для дипломного проєкту стали технічна література, документація технології, а також тематичні публікації у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- створити структуровану інструкцію, яка б охопила усі етапи проєктування електронних систем;
- спроєктувати пристрій (друковану плату), яка б отримувала дані з навколишнього середовища та відправляла їх за допомогою стільникового модему.

5.2. Вимоги до програмного забезпечення

- ОС MS Windows/Linux/Mac
- STM32CubeIDE
- KiCad 9.0
- Python 3.x + Matplotlib
- Node.js 22.x

5.3. Вимоги до апаратної частини

- 64-розрядний (x64) процесор із тактовою частотою не менше 2 ГГ
- Оперативної пам’яті не менше 4 ГБ
- Вільний простір на жорсткому диску щонайменше 5 Гб

6. ЕТАПИ РОЗРОБКИ

Етап виконання	Термін виконання
Затвердження теми роботи	04.04.2025
Вивчення та аналіз завдання	14.04.2025-04.05.2025
Розробка архітектури та загальної структури системи	21.04.2025-01.05.2025
Розробка структур окремих частин системи	02.05.2025-11.05.2025
Програмна реалізація системи	28.04.2025-11.05.2025
Виправлення помилок	11.05.2025-18.05.2025
Оформлення пояснювальної записки	24.04.2025-01.06.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати*»

Київ – 2025

ЗМІСТ

ЗМІСТ 1
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 2
ВСТУП 5
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 6
1.1. Актуальність та доцільність розробки 6
1.2. Огляд та приклади програмного забезпечення 8
ВИСНОВОК ДО РОЗДІЛУ 1 11
РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 12
2.1. Основи проєктування друкованих плат 12
2.2. Монтаж компонентів 17
2.3. Фізичні основи проєктування електричних схем 21
2.4. Програмування STM32 33
ВИСНОВОК ДО РОЗДІЛУ 2 47
РОЗДІЛ 3. РОЗРОБКА АВТОНОМНОЇ МЕТЕОСТАНЦІЇ НА БАЗІ STM32 48
3.1. Аналіз вимог та вибір датчиків 48
3.2. Алгоритм роботи мікроконтролеру 52
3.3. Проєктування електричної принципової схеми 53
3.4. Проєктування друкованої плати 55
3.5. Тестування розробленого пристрою 56
ВИСНОВОК ДО РОЗДІЛУ 3 60
РОЗДІЛ 4. РОЗРОБКА НАВЧАЛЬНО-МЕТОДИЧНОГО КОМПЛЕКСУ ДЛЯ ПРОЄКТУВАННЯ ЕЛЕКТРОННИХ СИСТЕМ 61
ВИСНОВКИ 83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACK	(Acknowledge) — підтвердження прийому (I2C)
ADC	(Analog-to-Digital Converter) — аналого-цифровий перетворювач
AHB1ENR	(AHB1 Enable Register) — регістр увімкнення периферії на шині AHB1
ARR	(Auto-Reload Register) — регістр автоперезавантаження таймера
CCR	(Capture/Compare Register) — регістр захоплення/порівняння (таймери)
CNT	(Counter) — лічильник таймера
CRC	(Cyclic Redundancy Check) — контрольна сума для перевірки цілісності
DAC	(Digital-to-Analog Converter) — цифро-аналоговий перетворювач
DMA	(Direct Memory Access) — прямий доступ до пам’яті
DR	(Data Register) — регістр даних
EXTI	(External Interrupt) — зовнішнє переривання
FPU	(Floating Point Unit) — блок обчислень з плаваючою комою
FSMC	(Flexible Static Memory Controller) — контролер зовнішньої статичної пам’яті
GPIO	(General Purpose Input/Output) — загального призначення ввід/вивід
GPIOD	(General Purpose Input/Output Port D) — порт загального призначення D
HAL	(Hardware Abstraction Layer) — рівень апаратної абстракції STM32
HSE	(High Speed External) — зовнішній кварцовий генератор
HSI	(High Speed Internal) — внутрішній RC-генератор
I2C	(Inter-Integrated Circuit) — послідовний протокол обміну даними
IDE	(Integrated Development Environment) — інтегроване середовище розробки
ISR	(Interrupt Service Routine) — обробник переривань

IWDG	(Independent Watchdog) — незалежний сторожовий таймер
MODER	(Mode Register) — регістр конфігурації режиму порту
MOSFET	(Metal Oxide Semiconductor Field Effect Transistor) — польовий транзистор з ізольованим затвором
NACK	(Not Acknowledge) — відмова у підтвердженні (I2C)
NSS	(Slave Select) — вибір підлеглого пристрою (SPI)
NVIC	(Nested Vectored Interrupt Controller) — контролер переривань
ODR	(Output Data Register) — регістр виводу даних
OLED	(Organic Light-Emitting Diode) — органічний світлодіодний дисплей
PLL	(Phase-Locked Loop) — система фазового автопідлаштування
RAM	(Random Access Memory) — оперативна пам’ять
RC	(Resistor-Capacitor) — RC-ланцюг або генератор
RCC	(Reset and Clock Control) — блок тактування і скидання STM32
RX	(Receive) — прийом даних (UART/SPI)
SCK	(Serial Clock) — тактовий сигнал (SPI)
SCL	(Serial Clock Line) — тактова лінія (I2C)
SD	(Secure Digital) — картка пам’яті SD
SDA	(Serial Data Line) — лінія передачі даних (I2C)
SDRAM	(Synchronous Dynamic Random Access Memory) — синхронна динамічна оперативна пам’ять.
SMT	(Surface-Mount Technology) — технологія поверхневого монтажу
SPI	(Serial Peripheral Interface) — послідовний периферійний інтерфейс
SR	(Status Register) — регістр стану
SRAM	(Static RAM) — статична оперативна пам’ять
STM32	Сімейство 32-бітних мікроконтролерів ARM Cortex від STMicroelectronics
THT	(Through-Hole Technology) — технологія наскрізного монтажу
TX	(Transmit) — передача даних (UART/SPI)

UART	(Universal Asynchronous Receiver-Transmitter) — послідовний інтерфейс
USART	(Universal Sync/Async Receiver-Transmitter) — розширений UART із підтримкою синхронізації
VBAT	(Battery Voltage) — живлення від батареї
VCAP	(Capacitor Voltage) — вихід стабілізації напруги ядра STM32
VDDA	(Analog Voltage) — живлення аналогової частини
VREF	(Voltage Reference) — опорна напруга для ADC
WWDG	(Window Watchdog) — сторожовий таймер з вікном

ВСТУП

Остання десятиліття супроводжувалося стрімким розвитком технології. Сьогодні все ширшого вжитку набувають вбудовані пристрої. Автоматизовані домашні системи, системи моніторингу навколишнього середовища – усі ці пристрої представляють собою електронні системи.

Створення таких систем вимагає глибокого технічного підґрунтя. Однак освоїти повний цикл розробки непросто, особливо враховуючи невелику кількість існуючих навчальних матеріалів та їх малодоступність, тому доцільно створити більш детальну інструкцію, яка б охопила усі етапи створення друкованих плат, а саме вибір та програмування датчиків, створення електричної принципової схеми та проєктування друкованої плати.

Перший розділ буде містити огляд сучасних рішень для проєктування електронних систем.

Другий розділ буде присвячений аналізу предметної області.

Третій розділ буде містити огляд та тестування розробленого пристрою, метеостанції на базі STM32.

Четвертий розділ буде містити інструкцію до розробки електричної схеми та друкованої плати.

В кінці роботи було отримано навчальний програмно-апаратний комплекс, створений на базі технологій, розглянутих у дипломній роботі.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Актуальність та доцільність розробки

Останні здобутки в сфері штучного інтелекту та машинного навчання призвели до стрімкої автоматизації промислових процесів. Автоматизація не оминула і розробку програмного забезпечення. Сьогодні написання коду, тестування і навіть проєктування усе частіше відбувається із використанням інтелектуальних інструментів. Це в свою чергу підвищило вимоги до фахівців в області вбудованих систем. Сучасний розробник має не лише знати мову програмування, а й розуміння процесів проєктування електронних систем.

Проєктування електронних систем можна розділити на наступні етапи:

1. визначення вимог до системи;
2. вибір та програмування датчиків;
3. створення принципової електричної схеми;
4. проєктування друкованої плати;
5. збірка та монтування компонентів;
6. тестування продукту.

Щоб усвідомити значення сучасних підходів, слід коротко розглянути еволюцію електроніки. У минулому електроніка проєктувалася та збиралася з інтегральних схем та дискретних компонентів, які з’єднувалися за допомогою провідників. Такі плати були громіздкими і ненадійними, їх було важко налагодити, а масове виробництво було ускладнено тим, що усі компоненти та провідники паялися вручну.

Сьогодні друковані плати представляють собою багатошарові пластини із діелектрика (найчастіше, склотекстоліту), на поверхні та всередині яких знаходяться провідникові доріжки для електричного з’єднання компонентів, а самі компоненти стали мініатюрними [1].

Не зважаючи на мініатюризацію компонентів, їх складність невпинно зростає, через що багатьом фахівцям доводиться самостійно вивчати

незліченні джерела, що відрізняються за складністю та тематикою. Це вимагає поєднання знань одразу із декількох галузей: схемотехніки, фізики, проєктування друкованих плат та програмування.

Наприклад, додаючи датчик до схеми, потрібно враховувати такі характеристики:

- протокол обміну даних (не лише через можливі обмеження мікроконтролеру, а і для фізичного підключення: наприклад, шини I2C потребують наявності підтягуючих резисторів, а підключення аналогових датчиків потребує екранування сигналу для забезпечення його цілісності);
- механічні параметри (розміри, спосіб монтажу);
- нюанси живлення, в тому числі струмове навантаження, допустимий діапазон напруги, потреба у додаткових стабілізаторах або конденсаторах;
- електроспоживання, особливо для автономних приладів;
- термічні характеристики (компоненти можуть вимагати додаткових термовідводів на платі або мати фіксований діапазон температурної експлуатації);
- наявність та/або необхідність калібрування;
- дотримання промислових стандартів;
- електромагнітна сумісність тощо.

Освоїти повний цикл розробки непросто, особливо враховуючи невелику кількість існуючих навчальних матеріалів, їх фрагментарність та малодоступність: більшість з них доволі фрагментарні або призначені для більш досвідчених розробників, тому доцільно створити більш детальну інструкцію, яка б поєднала теоретичні основи з прикладами з практики.

1.2. Огляд та приклади програмного забезпечення

Для проєктування електронних систем потрібно використовувати спеціальне програмне забезпечення, яке надає можливість щонайменше проєктувати електронні принципові схеми та проєктувати плати. З метою обрати найбільш підходяще програмне забезпечення, оглянемо найпопулярніші продукти.

Altium Designer [2], розроблений американською компанією Altium Limited, представляє собою один із найпотужніших та найбільш функціональних програмних комплексів для проєктування електронних систем промислової якості. Він включає в себе інструменти для створення електричних схем, проєктування друкованих плат та має інтеграції із дистриб'юторами компонентів. Водночас, він має високі системні вимоги (Intel® Core™ i7, 16 GB RAM) та високу ціну ліцензії – 4,235 $/рік і є складним до освоєння для новачків.

CircuitMaker [3] є іншим додатком від Altium Limited, який є безкоштовним та має досить широкий функціонал. Цей продукт є особливо популярним серед хобістів, однак доступний лише для Windows та має обов'язкову хмарну інтеграцію (що фактично означає, що усі проєкти залежать від третьої сторони), а також має певні функціональні обмеження і не так часто оновлюється.

EasyEDA [4] є веб-додатком для проєктування друкованих плат. Цей застосунок в першу чергу призначений для швидкого прототипування та має пряму інтеграцію з дистриб’юторами (замовлення компонентів можливо прямо із застосунку). Однак варто зазначити, що він потребує Інтернет-з’єднання, а також має обмеження по складності проєктів і слабші інструменти для перевірки схем. Ціна ліцензії – 240 $/рік.

Eagle [5], що наразі належить Autodesk, є ще одним популярним продуктом для проєктування електронних систем, однак його розробка наразі заморожена, а підтримка анонсована лише до 2026 року. З огляду на це, його

спільнота невпинно скорочується протягом останніх років, а сам редактор має репутацію дещо громіздкого та неінтуїтивного. Ціна підписки – 680 $/рік.

KiCad [6] є безкоштовним програмним забезпеченням з відкритим кодом. Має активну спільноту, велику кількість навчальних матеріалів. Є також українська локалізація, однак неповна. Інтерфейс вважається досить інтуїтивним, а наявність системи плагінів дозволяє додавати функціонал на будь-який смак, в тому числі й пряму інтеграцію з дистриб'юторами (в тому числі й генерацію виробничих файлів). Колись маловідомий, він отримав підтримку CERN і з того часу активно розвивається. KiCad вважається феноменом у світі програмного забезпечення з відкритим кодом, оскільки майже нічим не поступається платним аналогам.

Flux.ai [7] є сучасним редактором друкованих плат із інтегрованими інтелектуальними функціями для автоматизації процесів проєктування друкованих плат. Однак цей продукт ще знаходиться у розробці, через що деякі функції можуть бути недоступні. Ціна підписки - 348 $/рік.

Оскільки майже усі представлені вище програми мають достатній для навчання функціонал, а також безкоштовну версію для студентів, сформуємо декілька додаткових вимог:

- програма має бути доступна безкоштовно або за відносно невисоку ціну;
- функціонал програми має бути доступний повністю, без жодних обмежень;
- програма має мати невисокі системні вимоги і не вимагати підключення до Інтернету;
- програма має мати достатню кількість навчальних матеріалів та активну спільноту;
- програма має мати інтуїтивно зрозумілий інтерфейс, бажана також наявність української локалізації.

Проаналізувавши можливості та обмеження кожного програмного рішення та зважаючи на цілі роботи, найкращим варіантом для побудови навчально-методичного комплексу є KiCad 9.

Він є інтуїтивним та простим у освоєнні, а також має дуже активну спільноту, що гарантує його актуальність у найближчі роки, а також вважається певним стандартом для навчання.

ВИСНОВОК ДО РОЗДІЛУ 1

Оглянувши існуючі рішення для проєктування електронних систем, були отримані наступні висновки:

1) актуальність розробки навчально-методичного комплексу для проєктування електронних систем є беззаперечною, оскільки це відображає брак локалізованих та практично орієнтованих ресурсів українською мовою та сучасні вимоги до фахівців у сфері електроніки: сучасний інженер має володіти навичками усього циклу проєктування електронної системи;
2) рівень складності сучасної електроніки створює необхідність в комплексному підході, щоб студенти могли отримати практичні навички в проєктування пристрою від етапу вимог до розробки плати, а фокус на розробці друкованих плат у рамках навчально-методичного комплексу є обґрунтованим, оскільки друкована плата є основою усієї сучасної електроніки, забезпечуючи компактність та надійність з'єднань;
3) для проєктування електронних систем було обрано середовище KiCad 9.0, оскільки він має увесь необхідний функціонал, невисокі системні вимоги, є простим у використанні та є безкоштовним. Хоча інші альтернативи накшталт Altium Designer та EasyEDA є потужними, вони є суттєво складнішими для освоєння та мають більше апаратних вимог або потребують доступу до Інтернету;
4) доцільно реалізувати навчально-методичний комплекс на прикладі власного пристрою, а саме метеостанції, оскільки це підсилить практичну цінність матеріалу та дозволяє описати усі етапи проєктування, а саме вибір та програмування датчиків, створення принципової електричної схеми, проєктування друкованої плати, збірку та монтування компонентів та тестування продукту.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1. Основи проєктування друкованих плат

Як вже було згадано вище, друковані плати представляють собою багатошарові пластини із діелектрика, на поверхні та всередині яких знаходяться провідникові доріжки для електричного з'єднання компонентів [1].

Найчастіше, основою друкованої плати є мідна фольга з епоксидної смоли, армованої скловолокном (FR-4), але можуть також використовувати також, наприклад, ціанатний ефір або поліімід, якщо плати використовуються в умовах високих температур.

Друковані плати бувають декількох типів:

- односторонні, на таких платах компоненти розміщують лише з одного боку;
- двосторонні, на яких компоненти розміщують з обох боків;
- багатошарові, в яких провідні шари (металева фольга) розділені шарами із діелектрику – зазвичай препрегу. Препрег є клеєм, який все скріплює. Його виготовляють із скловолокнистої тканини, просоченої затверділою епоксидною смолою.

Для електричного з'єднання компонентів використовуються доріжки та отвори, які дозволяють проводити струм між шарами [8]. Передавати сигнали між шарами потрібно щоб уникнути конфлікту між доріжками. Так, наприклад, вважається за доцільно розміщувати шини живлення на окремому шарі (нижньому або внутрішньому), щоб уникнути конфлікту.

Доріжки можуть бути різної ширини (їх мінімальні розміри потрібно узгоджувати з розмірами на сайті виробника). Зазвичай ширина доріжок вимірюється в міліметрах або мілах, тисячних долях дюйму.

Ширина доріжок залежить від потреб сигналу: якщо потрібен захист від шуму або забезпечити пропускну здатність струму, доцільніше використовувати товщі доріжки.

Крім того, чим більше доріжки – тим швидше вони нагріваються, це особливо важливо для доріжок на внутрішніх шарах, оскільки вони мають нижчу теплопередачу.

Рекомендовано використовувати наступні значення ширини [8]:

— 0.5-1 мм для доріжок, що проводять струм низької напруги (до 1 А);
— для високопотужного струму або основної шини живлення: від 1.5 мм, рекомендовано від 2.5 мм;
— для цифрового сигналу потрібна доріжка товщиною від 0.2 мм, рекомендовано 0.5 мм;
— для аналогових сигналів та операційних підсилювачів – від 0.5 мм, рекомендовано – 0.8 мм.

При трасуванні доріжок потрібно також брати до уваги їхню топологію [8]:

— чим довша доріжка, тим більший опір та індуктивність, що погіршує сигнал;
— чим довша відстань між доріжками, тим вищий шанс пробою (електричне пробивання ізоляції) та так званого crosstalk'у: перехідного ємнісного зв'язку, коли сигнал з однієї доріжки наводиться на іншу; чим вища напруга — тим більша потрібна відстань, наприклад, для напруги 220 В мінімальна рекомендована відстань - 1.5 мм;
— у місцях, де змінюється напрямок, використовувати вигини під кутом 45 градусів, уникати прямих кутів, оскільки у таких кутах накопичуються заряди, що призводить електромагнітних завад і може спотворювати форму імпульсу;
— потрібно уникати паралельних доріжок або робити їх якомога коротшими, оскільки це може призвести до перехідного ємнісного зв'язку, а якщо уникнути такого розміщення неможливо, вони повинні бути на відстані мінімум у 3 рази більшій за ширину доріжки;
— для багатошарової плати вважається за доцільним один шар повністю віддати під землю;

Інший важливий елемент друкованих плат – отвори [9]. Вони бувають різних типів:

— наскрізні, тобто такі, що протягуються від верхнього шару до нижнього. Саме наскрізні отвори є найпоширенішими. Вони можуть бути металізовані або не металізовані (такі використовуються для фізичного монтування плати за допомогою гвинтів);

— сліпі перехідні отвори (blind viases), які протягуються від зовнішнього до внутрішнього шару;

— приховані перехідні отвори (buried viases), які протягуються між внутрішніми шарами;

— мікроотворами називають отвори діаметром менше 150 мікрон. Вони є особливо важливими елементами високощільних друкованих плат, особливо для тих, де важлива швидкість сигналу;

Отвори також можуть використовуватися для відводу тепла, зазвичай для цього використовують наскрізні або сліпі отвори.

Виготовлення друкованих плат відбувається у декілька етапів [10]. Враховуючи розміри сучасних електричних компонентів, для перенесення електричних схем на плату використовується лазерне пряме експонування.

Цей процес полягає у нанесенні на плату тонкого шару фоточутливої плівки – фоторезисту [10]. Цей матеріал містить хімічні речовини, що полімеризуються (затвердіють) під впливом ультрафіолетового випромінення.

Після цього плату розміщують під лазером, який формує бажаний малюнок електричної схеми. Освітлені ділянки схеми тверднуть (експонуються), тоді як неекспоновані залишаються м’якими. Наступним кроком є видалення неекспонованої міді лужним розчином – цей процес називають травленням [10].

Потім плати промивають водою для видалення залишкової плівки. Мідні доріжки також покриваються оксидом, щоб запобігти корозії та окисленню.

Після того, як усі шари плат готові, вони автоматизовано перевіряються оптично на предмет наявності дефектів поверхневого монтажу, ретельно

вирівнюються у необхідній конфігурації та ламінуються. Ламінуванням називається процес скріплення провідних шарів з препрегом: вони пресуються разом під впливом екстремальної температури та тиску [10].

Далі створюються перехідні отвори та паяються вивідні контакти. Свердління перехідних отворів є найдорожчим та найтривалішим процесом у виробництві друкованих плат, оскільки навіть незначна помилка може призвести до втрат у ефективності або ж плата може не працювати зовсім, тому для забезпечення точності цього процесу, використовуються рентгенівські промені.

Для друкованих плат застосовують два типи свердління: механічне та лазерне [9]. Механічні свердла менш точні і можуть просвердлити отвір до 6 міл, крім того вони також підвищують вимоги до матеріалів та товщини шарів. Лазерні свердла зазвичай використовуються для свердління перехідних отворів контрольованої глибини, можливий діаметр отвору – до 2 міл, однак цей процес є дорожчим і важчим, оскільки різні шари виготовлені з різних матеріалів, які в свою чергу мають різні оптичні властивості.

Після цього стінки отворів та поверхню плати металізують [10]. Це відбувається у декілька етапів.

На першому етапі усуваються можливі плями епоксидної смоли та сліди окислення. Крім того, внаслідок свердління можуть виникнути задирки (дрібні металеві виступи чи залишки матеріалу), від яких теж потрібно позбавитися, оскільки вони негативно впливають на електричні характеристики плати та можуть викликати короткі замикання [9].

Далі активація каталізатора: на стінки отворів наносять каталізатор, найчастіше на основі паладію, яка допомагає міді осісти.

Останнім кроком є хімічне осадження міді: плату занурюють у розчин міді, яка осідає на стінках отворів утворюючи шар товщиною 0,08-0,1 мікрон.

Наступним етапом є нанесення паяльної маски [10]. Маска не лише захищає доріжки від коротких замикань, запобігаючи потрапляння припою, а й відділяє місця для пайки (щоб припій прилипав тільки там, де потрібно); та захищає плату від бруду, пилу та вологи.

Для цього знову використовують лазерне пряме експонування: на поверхню наносять рідку фоточутливу маску, на якій знаходяться отвори та контакти для пайки, експонують її та очищають поверхню від неекспонованої плівки. Після цього плату нагрівають ще раз – в духовці або інфрачервоним випромінюванням, щоб маска затверділа [10].

Стандартним кольором паяльної маски є зелений, оскільки він не напружує очі (до впровадження автоматизованих систем контролю, усі плати перевірялися вручну).

Коли плата готова, контактні майданчики (де будуть паятися компоненти) залишаються відкритими, що призводить до окиснення (вже через кілька тижнів пайка стане проблемною), тому на поверхню наносять оздоблення (surface finish): ще одне покриття, яке захищає мідь від окиснення і полегшує пайку [10].

Існує декілька основних типів оздоблення:

- HASL (Hot Air Solder Leveling): плату занурюють в розплавлений олов'яно-свинцевий припій, надлишки здувають гарячим повітрям, також є варіанти безсвинцевих припоїв (екологічно чистий варіант) – найдешевший варіант, але не підходить для дрібних елементів;
- ENIG (Electroless Nickel Immersion Gold): на мідь наносять нікель, а зверху покривають тонким шаром золота;
- ENEPIG (Electroless Nickel, Electroless Palladium, Immersion Gold) представляє більш складне покриття: нікель, паладій, золото, що забезпечує стійкість плати;
- тверде золото: зони високого зносу (наприклад, контакти, які часто підключаються та відключаються) покривають нікелем та товстим, міцним золотом;
- імерсійне срібло: на мідь наносять тонкий шар срібла (срібло добре проводить струм і захищає мідь, але з часом темнішає);
- імерсійне олово: на мідь наносять тонкий шар олова, добре підходить для дрібних і багатошарових плат;

– OSP (Organic Solderability Preservative) – органічний консервант на водній основі, який забезпечує тимчасовий захист міді. Дуже дешевий, але недовговічний варіант.

Далі виконується шовкографія: так називається процес нанесення тексту й позначок (номери компонентів, логотипи, позначки для монтажу тощо) на плату.

Для цього струменевим принтером або через трафарет на плату наносять спеціально чорнило, а потім ще раз її запікають, щоб чорнило затверділо і не стерлося.

На останньому етапі відбувається електричне випробування плат. Цей тест перевіряє, чи усі доріжки і з’єднання на платі правильні: чи немає коротких замикань або обривів. Це важливо, оскільки після всіх етапів виробництва могли з’явитися дефекти.

Тестування відбувається за допомогою літаючих зондів: автоматизована машина з голками переміщається по платі і торкається потрібних точок та перевіряє наявність електричного контакту.

2.2. Монтаж компонентів

Після виготовлення друкованої плати відбувається етап монтажу компонентів. Існує кілька основних типів монтажу компонентів [11].

Наскрізний монтаж (THT, Through-Hole Technology) є традиційним методом для монтажу компонентів, який передбачає, що виводи компонентів вставляються в просвердлені отвори на друкованій платі, а потім припаюються до контактних майданчиків на протилежному боці.

Оскільки THT-компоненти кріпляться і механічно і пайкою, вони є дуже міцними, що забезпечує належну надійність для механічних навантажень, вібрації. Наскрізні з'єднання також забезпечують ефективне розсіювання тепла.

Водночас, такі компоненти часто вимагають більше місця на платі, а також унеможливлює монтування компонентів з обох боків плати (оскільки на

одному боці будуть виводи), що обмежує гнучкість проєктування електронних пристроїв.

Крім того, встановлення THT-компонентів неможливо повністю автоматизувати: вставка компоненту та паяння зазвичай потребує ручної роботи.

Типовими дефектами монтажу таких компонентів є холодна пайка або недопайка (було використано недостатньо припою або нагрів був недостатньо) та обриви (механічне пошкодження виводів або доріжок).

SMT – Surface Mount Technology або технологія поверхневого монтажу, яка передбачає монтаж компонентів на поверхню плати, є сучасним стандартом для монтажу компонентів на платі [11]. Такі компоненти називаються SMD-компонентами (Sufrace Mount Device). Цей метод революціонізував електроніку, оскільки дозволив значно збільшити щільність елементів на платі.

Монтаж відбувається у декілька кроків: на визначені місця на платі наносять паяльну пасту, після чого компоненти розміщаються машиною. Після цього плата проходить етап пайки оплавленням (reflow soldering): її поміщають у піч, де вона поступово нагрівається [10]. Завдяки цьому паяльна паста розплавляється і утворює міцні сплави між виводами компоненту і контактним майданчиком.

SMD-компоненти мають коротші шляхи з'єднання, це також мінімізує спотворення сигналу. Оскільки монтаж SMD-компонентів відбувається автоматизовано, це пришвидшує і здешевлює виробництво. Саме автоматизація дозволяє мініатюризувати компоненти: машина здатна точно встановити крихітний компонент на плату.

Однак, такий розмір є і мінусом: процес розміщення вимагає точного контролю нанесення паяльної пасти та розміщення компоненту, що вимагає складного обладнання і унеможливлює ручний ремонт. Крім того, хоча сам процес складання SMD-компонентів дешевший, самі компоненти зазвичай коштують дорожче.

Для створення сучасних пристроїв часто використовують змішану збірку (mixed assebmly), коли на одній платі використовуються компоненти обох типів [10]. Це пов’язано з тим, що більші та потужніші компоненти накшталт роз’ємів та котушок потребують більшої механічної міцності, тоді як компоненти дрібніше – резистори, конденсатори – монтуються на поверхню.

Для поверхневого монтажу типові такі дефекти:

- містки припою: надлишок припою може може призвести до короткого замикання;
- недостатнє змочування (змочування називається процес, коли паяльна паста стає розплавленою та здатною прикріплюватися до виводу компонента та до друкованої плати): коли паяльна паста не повністю покриває контактну площадку;
- тріщини в припої через термічний стрес (піч потребує поступового нагріву та охолодження);
- зміщення компонентів;
- ефект “надгробка” (tombstoning, також відомий як ефект Мангеттена або ефект крокодилу) – коли один кінець компонента відривається від контактної площадки друкованої плати під час процесу паяння.

2.2.1. Пайка компонентів

Відрізняються також і типи пайки компонентів. Він залежить від типу монтажу та габаритів плати.

Для прототипів може використовуватися ручна пайка [12]. Для ручної пайки потрібен паяльник чи паяльна станція (для паяння плат підійде паяльник невисокої потужності), флюс (речовина для видалення окисів і шлаків із розплавленого металу для запобігання окиснення; найпоширенішим флюсом є каніфоль, варто також зауважити, що в більшості популярних припоїв флюс є в складі), припій (припої поділяються на свинцеві та безсвинцеві; другі, очевидно, є безпечнішими, однак набули значної поширеності лише

нещодавно, оскільки свинцеві припої мають нижчу температуру плавлення: для більшості олов'яно-свинцевих припоїв температура коливається в діапазоні від 180 до 190 ° С) та стрічки для зняття припою.

Перед початком паяння потрібно залудити жало паяльника [12]. Лудінням називають процес покриття металевої поверхні тонким шаром припою. Для цього потрібно розігрітий інструмент занурити у каніфоль та покрити припоєм: таким чином олово рівномірно покриє наконечник, що убереже його від окиснення та покращить адгезію припою (адгезією називають здатність двох різних матеріалів прилипати один до одного на молекулярному рівні). Жало треба очищати від залишків припою після кожного паяння.

Перед початком процесу пайки потрібно підготувати компоненти: усі деталі мають бути очищені від пилу та бруду [12].

Після цього, треба встановити та зафіксувати компоненти, а на їх поверхню треба нанести трохи флюсу.

Перед нанесенням припою треба впевнитися, що паяльник досяг правильної температури: холодна пайка зробить шви неякісними та нестійкими, а від перегрітого жало погіршяться характеристики припою. Щодо нанесення припою, є два принципових підходи:

- нанести припой на жало;
- нанести припой на поверхню компонентів [12].

Кількість припою обирається на око, однак його не потрібно бути забагато, щоб він не розтікся, а замалої кількості буде просто недостатньо, оскільки олово просто не заповнить всі мікрощілини між поверхнями.

Розжарений інструмент потрібно тримати в зоні паяння не більше декількох секунд.

У промисловості зазвичай використовуються:

- хвильова пайка (wave soldering);
- пайка оплавленням (reflow soldering);
- пайка гарячим повітрям (hot air soldering);
- пайка лазером (laser soldering).

Хвильова пайка застосовується переважно для THT-компонентів [10]. Для цього зібрану плату покривають флюсом (щоб уникнути окиснення) і проводять над хвилею розплавленого припою (плату проводять над ванною із розплавленим припоєм, де спеціальні насоси або лопаті формують і постійно підтримують струмінь припою, який утворює хвилю). Таке паяння є складнішим технологічно, оскільки потрібно враховувати розмір контактних майданчиків, орієнтацію плати та безліч інших факторів, але він може використовуватися для масового виробництва.

Пайка оплавленням є найпоширенішим індустріальним методом паяння. Для цього друкована плата попередньо нагрівається (для запобігання термічного стресу) і поміщається в піч для оплавлення, де гаряче повітря плавить пасту для утворення паяних з'єднань [10]. Цей метод вважається найпростішим.

При пайці гарячим повітрям, розплавлення припою досягається за рахунок розігрітого потоку повітря [10]. Фен подає струмінь гарячого повітря на місце пайку, в результаті чого припій або паяльна паска плавиться і компонент можна встановити або зняти пінцетом. Найкраще цей метод підходить для ремонту або заміни компонентів.

При пайці лазером, лазер фокусується на місце пайки і передає сфокусоване тепло на невелику ділянку. Цей метод ідеально підходить для чутливих елементів та використовується для високощільних плат та мікроелектроніки.

2.3. Фізичні основи проєктування електричних схем

Проєктування електричних схем є важливим етапом проєктування електронних систем. Однак щоб складати будь-які електричні схеми, потрібно згадати деякі основні положення з фізики.

Електричний струм – упорядкований рух електрично заряджених частинок. Кількісною характеристикою струму є сила струму, вимірюється в амперах та відповідає кількості заряду, що проходить крізь переріз провідника за певний проміжок часу. Ми вважаємо, що струм направлений від

позитивного полюсу до негативного, хоча насправді, звісно, провідниками струму в металах є електрони, а вони мають негативний заряд, тобто рухаються від негативного полюсу до позитивного.

Напруга – різниця електричних потенціалів між двома точками. Напругу також називають електричним тиском, оскільки її можна уявити як силу, що штовхає електрони крізь провідник, подібно до того, як тиск змушує воду рухатися по трубі. Позначається літерою U та вимірюється у вольтах. Напруга між точками може бути спричинена накопиченням електричного заряду (наприклад, конденсатором) або електрорушійною силою (джерелом енергії, наприклад, батарейкою або генератором).

Електричний опір характеризує властивість провідника створювати протидію проходженню електричного струму. Опір позначається літерою R та вимірюється у омах.

Ці величини пов’язані законом Ома: сила струму у провіднику між двома точками прямо пропорційна напрузі та протилежно пропорційна опору.

Компоненти в електричному колі можуть бути з’єднані послідовно або паралельно, і це впливає на розподіл струму та напруги.

При послідовному з’єднанні всі елементи підключені один за одним — струм у них однаковий, а напруга розподіляється між ними залежно від опору.

При паралельному з’єднанні обидва виводи кожного елемента підключені до одних і тих самих точок — тому напруга на всіх однакова, а струм розподіляється залежно від опору кожного елемента.

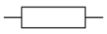
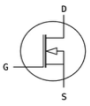
Уявити це можна через аналогію з рухом електричних зарядів: у послідовному з’єднанні один і той самий потік зарядів проходить через усі елементи, витрачаючи енергію на кожному опорі. У паралельному з’єднанні заряди мають можливість розділитися між кількома шляхами, де кожен шлях протікає під однаковою різницею потенціалів.

Варто зауважити, що електрони рухаються дуже повільно, міліметри або сантиметри за секунду, однак струм з’являється майже одночасно. Це пов’язано з поняттям електричного поля: воно моментально (майже зі

швидкістю світла) поширюється по всьому провіднику. Це поле створює і силу, яка змушує всі вільні електрони в провіднику рухатися.

Розглянемо деякі з найпоширеніших електричних елементів та їх позначення (табл. 2.1).

Таблиця 2.1 – Найпоширеніші електричні елементи

Елемент електричного кола	**Умовне позначення**
Резистор	
Конденсатор	
Котушка індуктивності	
Гальванічний елемент	
Діод	
Операційний підсилювач	
P-MOSFET	
N-MOSFET	

Резистор — це пасивний елемент, який обмежує або регулює струм у колі. Він чинить опір руху електронів, що призводить до зниження струму та перетворення частини електричної енергії в тепло. Наприклад, резистор можна поставити перед світлодіодом, щоб обмежити струм і не знищити діод.

Конденсатор представляє собою два провідники, розділені діелектриком. Він накопичує електричний заряд і енергію в електричному полі. При підключенні до джерела напруги накопичує заряд, а потім може його віддати. Конденсатор може діяти як фільтр або накопичувач. Наприклад, коли

живлення "просідає" (наприклад, при спалаху споживання), конденсатор короткочасно віддає заряд, таким чином інша цифрова логіка не постраждає

Котушка накопичує енергію в магнітному полі при протіканні струму. Протікання струму створює магнітне поле. При зміні струму котушка протидіє цій зміні, утворюючи електрорушійну силу (ЕРС). Наприклад, у стабілізаторах живлення котушка накопичує енергію, коли транзистор відкритий, а потім віддає її, коли він закритий — це дозволяє згладжувати напругу або перетворювати її

Гальванічний елемент забезпечує постійне джерело електричної енергії. основі лежить хімічна реакція, яка створює різницю потенціалів (напругу) між двома електродами.

Діод пропускає електричний струм тільки в одному напрямку. Працює за принципом p-n переходу, що має односторонню провідність: коли анод має вищий потенціал за катод — струм проходить, в іншому напрямку — блокується. Діоди часто використовуються для захисту мікроконтролера — наприклад, у випадку переплутаної полярності на вході живлення, діод не дозволить зворотному струму пошкодити схему.

Також діоди використовуються у випрямлячах для перетворення змінного струму в постійний або в захисті від імпульсів — наприклад, паралельно до котушки реле, щоб погасити зворотню напругу при вимкненні навантаження.

Операційний підсилювач представляє собою універсальний аналоговий елемент, який використовується для підсилення сигналів, порівняння напруг, фільтрації та перетворенні аналогових сигналів у цифрові.

Операційний підсилювач має два входи: неінвертуючий (позначається плюсом, вхідний сигнал), інвертуючий (зворотній зв'язок або опорна напруга) та вихід.

Цей компонент є основою *компараторів* (операційний підсилювач без негативного зворотного зв'язку називається компаратором, для цього мінус операційного підсилювача підключається через резистор до землі), *фільтрів*, *підсилювачів струму та напруги*.

MOSFET (Metal–Oxide–Semiconductor Field-Effect Transistor) — це польовий транзистор з ізольованим затвором [13]. Транзистори є ключовими елементами сучасної електроніки. Вони представляють собою напівпровідникові елементи, які можуть перемикати та підсилювати сигнали.

Транзистори працюють на основі p-n переходу, яким називають межу між двома ділянками напівпровідника з різною провідністю. Літерою "P" позначають частину, де більше позитивних носіїв ("дірок", так називають місця, де не вистачає електронів), а "N" – де більше негативних (електронів). У місті поєднання двох таких матеріалів електрони намагаються переміститися у місця дірок, а дірки – на місце електронів. Через певний час поблизу межі утворюється зона виснаження, оскільки там не залишається вільних носіїв заряду.

Як результат, якщо подати напругу у правильному напрямку, то струм проходить (напруга буде штовхати електрони і дірки один до одного) і навпаки (зона виснаження буде розтягуватися).

MOFSET має три виводи:

- Gate (G) – затвор, який керує роботою;
- Drain (D) – сток, куди тече струм;
- Source (S) – витік, звідки надходить струм.

Принцип роботи MOSFET полягає у тому, що коли напруга на затворі недостатня, канал закритий і струм не тече.

У N-канальному MOSFET струм тече від стоку до витоку. У P-канальному MOFSET струм тече навпаки: від витоку до стоку.

Будь який мікроконтролер представляє собою програмований електронний пристрій, який має набір виводів (пінів), які працюють в різних режимах.

Піни мікроконтролера — це входи транзисторів, які дуже чутливі до випадкових напруг, залишкових зарядів або електромагнітних наводок (кожен пін підключений до затвору MOSFET, затвор є вхідним вузлом із дуже високим опором, тому на нього можуть впливати навіть найменші напруги, і

навіть простий провід може діяти як антена). Це призведе до непередбачуваної поведінки входу.

Якщо пін "висить", напруга на затворі транзистора стрибає. Варто зазначити, що для цифрової логіки STM32 “0” це живлення від 0 до 0.8 В, а “1” – вище. Тому мікроконтролер буде бачити хаотичні переходи від “0” до “1”. Для розв’язання цієї проблеми використовуються підтягуючі (pull-up) або стягуючі (pull-down) резистори [14]. Підтягуючі резистори з'єднують вхідний провідник із позитивним живленням а стягуючі — із землею.

Якщо на вхід поставити підтягуючий резистор (рис. 2.1), то при відсутності активного сигналу струм буде слабо тягнути електрони до VDD (наприклад, 3.3 В). На вході накопичиться позитивний потенціал, і рівень буде сприйматись як логічна “1”.

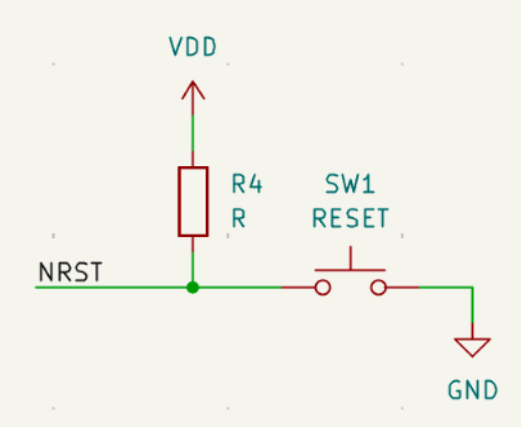


Рисунок 2.1 – Приклад підтягуючого резистору

Якщо поставити стягуючий резистор (рис. 2.2), то він буде тягнути електрони до GND, поки нічого не підключено. Вхід буде триматись у низькому стані — логічна “0”.

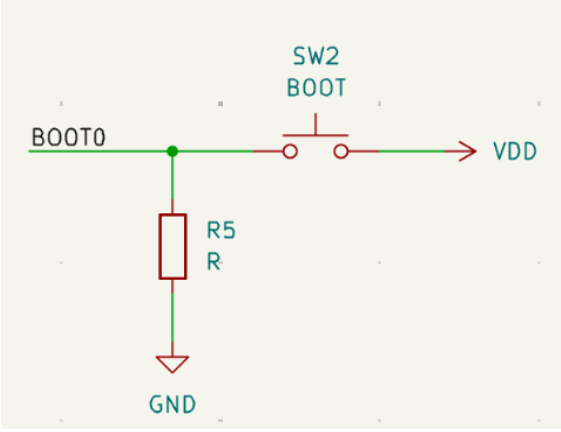


Рисунок 2.2 – Приклад стягуючого резистору

Підтягування або стягування потрібно коли пін налаштовано як вхід (GPIO Input), коли лінія може залишатися неактивною (наприклад, SDA/SCL). Типові приклади:

- кнопка (GPIO Input) потребує pull-up або pull-down, залежно від того, в який стан має переходити лінія, коли вона натиснута;
- SDA/SCL потребує pull-up резисторів, оскільки передача даних відбувається шляхом стягування в "0";
- ADC може потребувати pull-down, якщо аналоговий вхід може бути неактивним;
- USART RX може потребувати pull-up, якщо передавач може бути відключеним;

Іншою важливою властивістю, яку потрібно мати на увазі при проєктуванні електричних схем є електромагнітна сумісність, яка характеризує здатність компонентів функціонувати в умовах радіозавад та не створювати радіозавад для інших. Це особливо важливо враховувати, якщо використовуються аналогові датчики або високошвидкісні інтерфейси. Для цього потрібно:

- використовувати суцільні площини для заземлення (для забезпечення зворотнього шляху для струму з низьким імпедансом);
- екранувати чутливі ділянки та живлення (за допомогою LC-фільтрів або феритового намиста: ферит є магнітним матеріалом, який на високих частотах втрачає магнітні властивості, в результаті чого підвищується його опір, який призводить до поглинання енергії поля у вигляді тепла і погашенню шуму);
- розділяти цифрові та аналогові доріжки.

Інколи має сенс враховувати температурні коефіцієнти компонентів, а також деякі їхні властивості, зокрема змінюється опір резисторів та ємність конденсаторів.

Температурні характеристики описуються температурним коефіцієнтом (TCR), який вимірюється в частках на мільйон на градус (ppm/°C). Для елементів із високою точністю, наприклад, АЦП та фільтрів, слід

застосовувати резистор з низьким температурним коефіцієнтом (< 50 ppm/°C) і конденсатори із стабільними діелектриками (наприклад, C0G або NP0).

На високих частотах також може виникнути скін-ефект, коли змінний струм тече лише по поверхні провідника, через що збільшується опір (оскільки зменшується площа провідника). На практиці це означає, що якщо період сигналу менше 20 нс, навіть короткі доріжки поводять себе як електричні лінії передач, а не як провідник, тобто сигнал не миттєво доходить до кінця доріжки, а отже частково відбиватися назад.

2.3.1. Реалізація автономного живлення

Одним із найважливіших етапів проєктування електронних систем є створення автономного живлення, оскільки більшість вбудованих пристроїв мають працювати без постійного живлення. Для цього зазвичай використовують батарейку або акумулятор.

Однак лише одної батарейки недостатньо. По-перше, бажано було б захистити батарейку від неправильної полярності (переполюсування), оскільки ані мікроконтролер, ані мікросхеми найчастіше не захищені від зворотної напруги. Це може призвести до:

- вибуху або роздуванню конденсаторів, оскільки вони не можуть працювати у зворотній полярності;
- спаленню мікросхем, які не мають окремого захисту: струм піде паразитними шляхами (незаплановані шляхи для струму), які мають високий опір і швидко перегріваються, через що з'являються внутрішні короткі замикання і кристал (так називають тонку кремнієву пластину, на якій розміщені мікроскопічні елементи розмірами мікронів і яка і реалізує основну логіку пристрою) плавиться або пробивається, що призводить до пошкодження мікросхеми назавжди;
- нарешті, сама батарея може нагріватися або розряджатися струмом короткого замикання, що може спричинити вибух або загоряння.

Це можна зробити декількома шляхами. Найпростішим варіантом буде додати діод у прямому включенні між батареєю та живленням схеми (якщо полярність буде неправильною, то струм не пройде), однак це викликає падіння напруги до 0.7 В (на діоді Шотткі – 0.3 В), що може бути критичним при низьковольтному живленні.

Більш сучасна альтернатива, P-канальний MOSFET у зворотному включенні із затвором, під'єднаним до землі через резистор, майже немає опору.

По-друге, мікроконтролер STM32 витримує максимум 3.6 В, а акумулятор може мати відмінну номінальну напругу, через що пряме підключення призведе до пошкодження мікросхеми.

Для захисту мікросхеми можна використати лінійний стабілізатор з малим падінням напруги (low-dropout regulator), наприклад, MCP1700-3.3 [15], який працює до 6 В і стабільно видає 3.3 В (рис. 2.3).

Він працює наступним чином: всередині мікросхеми є операційний підсилювач, який порівнює вихідну напругу з внутрішнім джерелом опорної напруги (так званий bandgap reference), який керує P-канальним MOSFET, що змінює струм (якщо вихід менший, то підсилювач знижує напругу на затворі і MOSFET відкривається сильніше, що призводить до підвищенню напруги, і навпаки). На вході та виході потрібні керамічні конденсатори щонайменше 1 мкФ для стабільності струму (для згладження шуму та стрибків напруги).

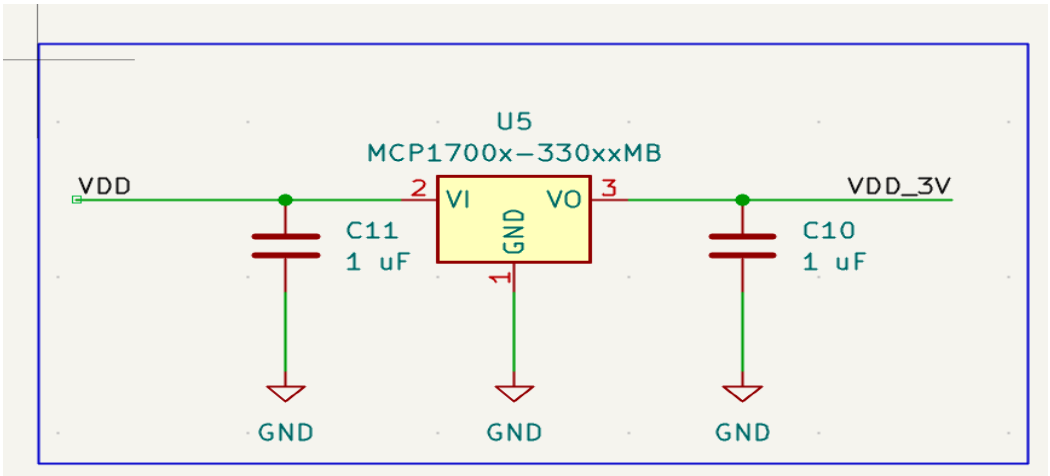


Рисунок 2.3 – Приклад підключення MCP1700-3.3 [6]

Однак нам також може знадобитися окреме живлення 5 В для датчиків. Для того, щоб підняти напругу використовують підвищувальний DC-DC перетворювач (dc-dc step-up converter), наприклад, ETA1096E8A [16] (рис. 2.4). Усередині мікросхеми швидко перемикається MOSFET, це створює накопичення енергії в котушці, яка потім різко виштовхується.

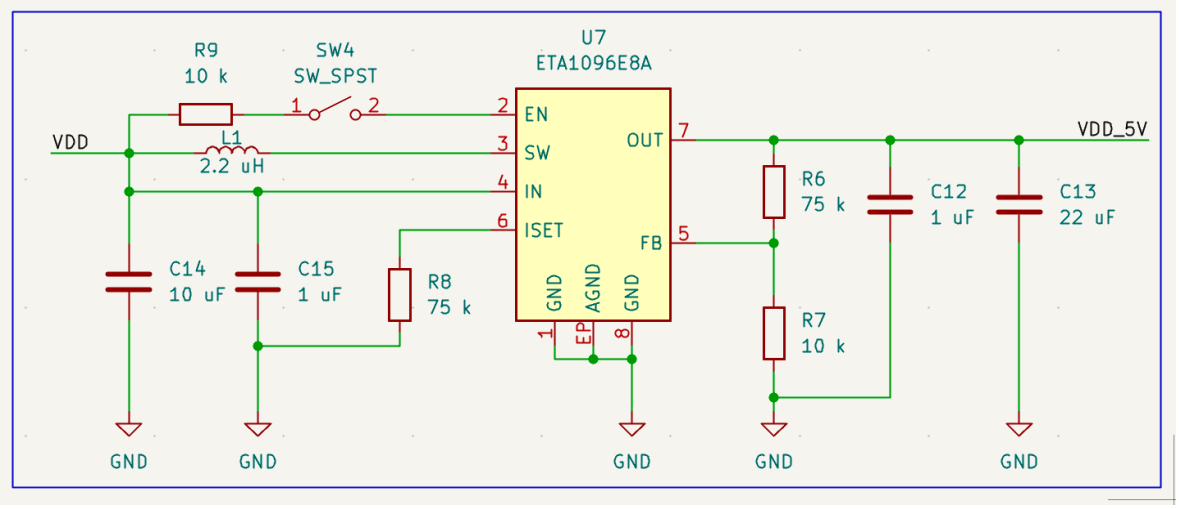


Рисунок 2.4 – Приклад підключення ETA1096E8A [6]

Насправді застосування підвищувального перетворювача має сенс навіть якщо немає датчиків, оскільки він дозволяє живити пристрій при зниженні напруги до 2.8 В, тоді як лінійний стабілізатор працює лише до 3.4 В, що означає, що із підсилювачем пристрій може використати до 90-95% ємності батареї, а лише із лінійним стабілізатором лише 70-75%.

Додатково можна забезпечити захист від надмірного розряду, а якщо мікросхема має зарядку – то і від перезаряду, наприклад, із використанням контролеру захисту DW01A [17], оскільки напруга Li-Ion акумулятора зазвичай змінюється в межах від 2.7 до 4.2 В. Якщо батарея розряджається нижче 2.5 В, вона може втратити ємність назавжди, якщо заряджає вище 4.3 В, то батарея може перегрітися та вибухнути.

Такі мікросхеми зазвичай працюють за допомогою MOSFET'ів (зазвичай, подвійний N-канальні MOSFET), які закриваються якщо напруга зависока або занизька.

2.3.2. Важливі нюанси живлення STM32

STM32 має багаторівневу архітектуру живлення [18]. Для наочності розглянемо символ компоненту STM32F407VGTx у середовищі KiCad (рис. 2.5):

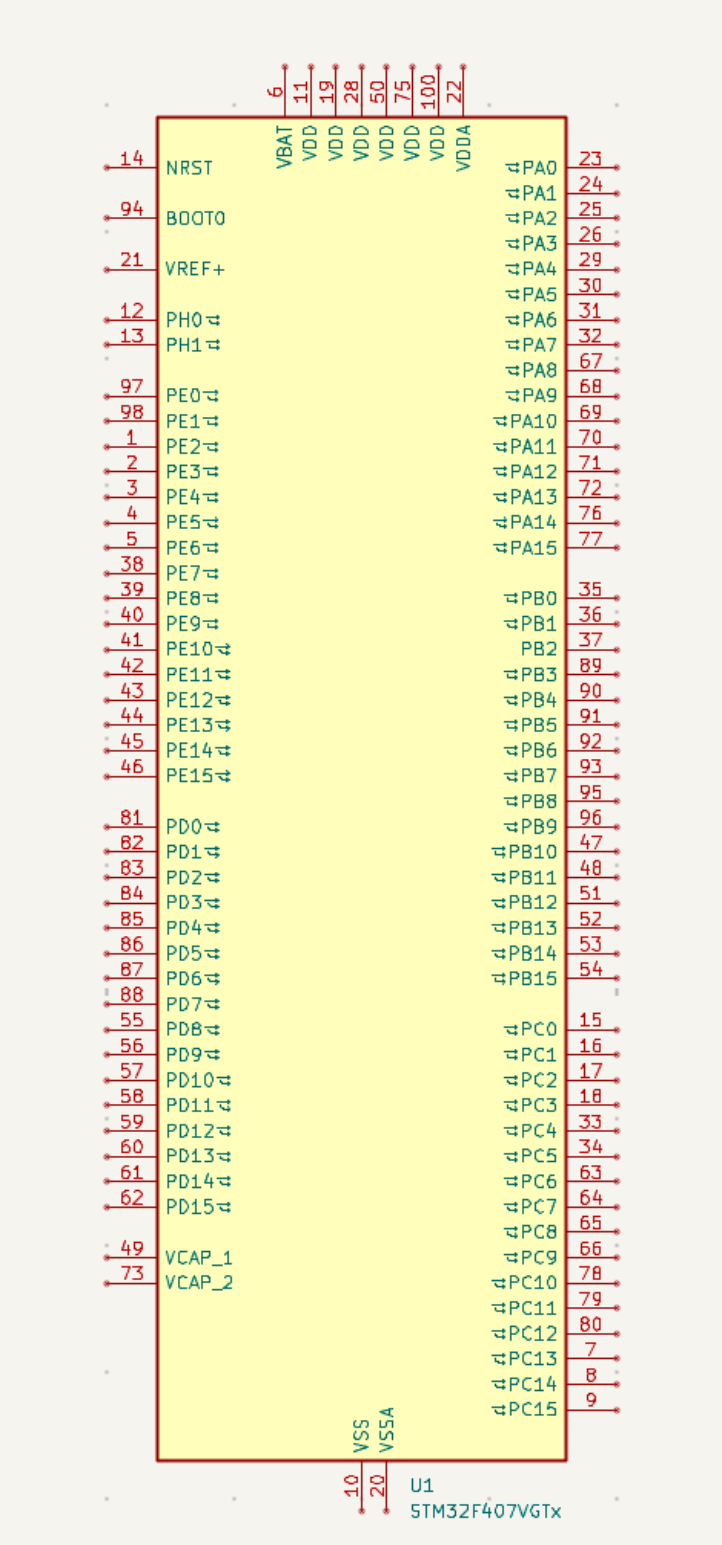


Рисунок 2.5 – Символ STM32F407VGTx у середовищі KiCad [6]

Схема живлення виглядає наступним чином:

– VDD/VSS — основні лінії живлення цифрової логіки (подаються 1.8-3.6 В і земля відповідно). Кожну пару VDD/VSS необхідно шунтувати окремим керамічним конденсатором 100 нФ, розміщеним якомога ближче до виводів. Конденсатори необхідні для розв'язування (decoupling) – так називається метод ізоляції частини схеми для зменшення шумів та завад. Такі конденсатори називають розв'язувальними та мають бути розміщені максимально близько до пінів живлення. Бажано також підключити один із входів VDD через конденсатор 4.7 мкФ;
– VDDA/VSSA - живлення аналогової частини (ADC, DAC, PLL, Reset-блоки). Рекомендовано підключати їх окремо для зменшення шумів, особливо якщо використовується АЦП;
– VBAT представляє собою резервне живлення для збереження даних в backup RAM та роботи RTC (real time clock) при відключенні основного живлення. Підключати VBAT необхідно тільки якщо потрібна робота RTC або backup RAM при відсутності основного живлення, інакше VBAT можна залишити не підключеним;
– VREF - опорна напруга для АЦП і ЦАП. В простих схемах її підключають безпосередньо до VDDA. Якщо потрібна підвищена точність, можна подавати стабілізовану напругу з окремого джерела. Якщо точність не критична, вивід VREF можна залишити непідключеним — мікроконтролер сам з'єднає його з VDDA;
– VCAP - виводи для підключення конденсаторів живлення внутрішнього стабілізатора ядра (STM32 має внутрішній лінійний стабілізатор для зниження 3.3 В до 1.2 В, напруги для живлення ядра). До кожного потрібно під'єднати керамічний конденсатор на 2.2 мкФ.

STM32 також має декілька різних режимів роботи:

1. Sleep mode: ядро зупиняється, але таймери та периферія продовжує працювати, пробудження відбувається при будь-якому перериванню.

2. Stop mode: забезпечує найнижче енергоспоживання при збереженні вмісту регістрів та оперативної пам'яті (SRAM), можна пробудити зовнішнім перериванням.
3. Standup mode: найменше енергоспоживання, однак SRAM та регістри втрачаються, окрім backup-домену. Вихід з такого режиму можливий при зовнішньому скиданні (пін NRST), скиданні через сторожовий таймер (IWDG), фронт сигналу на піні WKUP, події від RTC. Цей режим не підтримується якщо вбудований стабілізатор вимкнений або VCAP живиться від зовнішнього джерела

Режими роботи мікроконтролеру можуть бути змінені програмно за допомогою відповідних HAL-функцій:

— *HAL_PWR_EnterSLEEPMode*
— *HAL_PWR_EnterSTOPMode*
— *HAL_PWR_EnterSTANDBYMode*

Також STM32 підтримує два режими масштабування напруги (voltage scaling) для ядра мікроконтролера.

Ядро STM32 живиться напругою 1.2 В та може працювати із максимальною частотою до 168 МГц. Однак є другий рівень напруги: 1.14 В, максимальна частота – 144 МГц.

Щоб перейти у другий режим, можна використати функцію *__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2*).

При перемиканні у другий режим зниження, споживання знижується на 15-30 % у активному режимі на тій самій частоті. Споживання також зменшиться при зниженні частоти.

2.4. Програмування STM32

Мікроконтролери STM32 представляють собою пристрої на базі ядра ARM Cortex-M. Вони мають наступні основні функціональні блоки:

1. RCC (Reset and Clock Control) – керує тактуванням усіх підсистем. Тактуванням називають процес подачі регулярного сигналу (який ще називають тактовим імпульсом), який визначає швидкість роботи

мікроконтролеру (всі цифрові блоки працюють синхронно). Джерелом таких сигналів у STM32 є генератори:

— HSI (High-Speed Internal) – внутрішній RC-осцилятор 16 МГц;

— HSE (High-Speed External) – кварц або генератор 4-26 МГц;

— PLL – система, що множить частоту HSI/HSE до бажаної (максимум – 168 МГц).

Для стабільного тактування використовують кварцевий резонатор: кристал кремнезему, який має п'єзоелектричні властивості, тобто здатен змінювати свою форму при подачі напруги, і навпаки.

При подачі напруги кристал стягується або розтягується, після чого він намагається повернутися у вихідну форму, через що починає коливатися самостійно. Якщо частота збігається з власною резонансною частотою кристалу, він починає резонувати.

Ці коливання вловлює електронний контур і перетворює у стабільний сигнал.

RC-генератор працює наступним чином: він представляє собою електронний осцилятор, який використовує резистор та конденсатор. Конденсатор заряджається через резистор, що призводить до зростання напруги. Коли напруга досягає певного порогу, вмикається транзистор, який розряджає конденсатор. Після цього цикл повторюється. Цей механізм створює періодичні імпульси, але має доволі низьку точність (±1–3%).

2. GPIO (General Purpose I/O) – універсальні порти вводу-виводу (можуть, наприклад, прочитати стан кнопки або передати сигнал на світлодіод чи реле).
3. USART/UART/I2C/SPI – серійні інтерфейси для обміну даними із периферією.
4. TIM (таймери) – застосовуються для вимірювання часу, генерації затримок, подій, ШИМ-сигналів. У STM32 таймери представляють собою апаратні лічильники. Вони відраховують імпульси від тактового сигналу і виконують дії, коли лічильник досягає певного значення.

Таймер складається з наступних елементів:

— лічильника (counter, CNT), який рахує такти;

— поріг переповнення (ARR): коли CNT = ARR, CNT скидається у 0;

— дільника тактової частоти (prescaler, PSC);

— CCR (capture/compare register), який захоплює сигнал (зчитує значення CNT, коли на вхід подано імпульс) або порівнює лічильник (генерує подію або змінює стан виходу коли CNT = CCR);

— DIER (DMA/Interrupt Enable Register), який вмикає переривання;

— SR (Status Register) - прапорці подій;

— CR (control register), який визначає режим роботи.

STM32 підтримує наступні режими роботи таймерів:

— basic – простий лічильник з переповненням;

— PWM (Pulse Width Modulation) – ШІМ, таймер змінює рівень сигналу при досягненні CCR;

— input capture, який вимірює період сигналу (таймер захоплює значення лічильника коли на вхід приходить імпульс);

— output compare, який генерує події, коли CNT = CCR;

— one-pulse mode, який генерує одиничний імпульс при запуску;

— slave mode для синхронізації з іншим таймером.

5. ADC/DAC – для аналогових перетворень.
6. DMA (Direct Memory Access) – забезпечує прямий обмін даних між периферією та пам'яттю без участі ядра.
7. NVIC (Nested Vector Interrupt Controller) – контролер переривань, що підтримує пріоритетну обробку події. Сумарно підтримує 16 рівнів пріоритету, 82 каналів. NVIC працює за наступним алгоритмом: коли відбувається певна подія, змінюється біт у регістрі статусу (наприклад, при переповненні таймеру або коли UART отримав байт). Якщо переривання увімкнені, то NVIC передає керування ядру, яке зберігає контекст, призупиняє поточний код та переходить до ISR (Interrupt

Service Routine) – функції обробки. Якщо під час одного ISR виникає інше переривання з вищим пріоритетом — воно перериває поточне ISR.

8. EXTI (External Interrupt) – модуль, що дозволяє обробляти сигнали із зовнішніх пінів (GPIO) як переривання або події. EXTI лише чекає, поки не виконається умова та відправляє запит до NVIC.
9. IWDG (Independent Watchdog) та WWDG (Window Watchdog) – сторожові таймери, які є апаратними механізмами безпеки: вони скидають мікроконтролер, якщо програма зависла. IWDG має незалежне живлення та працює навіть у режимах stop та standby. Він має внутрішній лічильник, який з певною частотою зменшується. Якщо програма його не встигла перезавантажити, то виконується автоматичний рестарт системи. WWDG є більш жорстким варіантом сторожового таймеру. Він має вікно часу: проміжок, коли дозволено перезавантажити таймер. Мікроконтролер перезавантажиться, якщо програма перезавантажить таймер занадто рано або занадто пізно. Це гарантує, що програма не лише працює, а й працює із правильною періодичністю. Це особливо важливо у критичних системах, де не можна допустити жодного зависання.
10. RTC (Real Time Clock) – апаратний годинник, який працює незалежно від мікроконтролера за допомогою лічильника та тактового генератору.
11. FPU – апаратний модуль, який виконує операції над числами з плаваючою крапкою. Без FPU, наприклад, ділення чисел із плаваючою крапкою буде відбуватися програмно (по байтам), що займає 80-150 тактів часу.
12. Flash-пам’ять – енергонезалежна пам'ять, у якій зберігається програма, константи тощо. Така пам'ять працює на транзисторах із плаваючим затвором (floating gate). Запис даних базується на накопиченні заряду на ізольованій ділянці, яка здатна зберігати стан без живлення.

Програмування STM32 доступне на різних рівнях абстракцій: від роботи із регістрами до високорівневих бібліотек та використання інтелектуальних інструментів для автоматичної генерації коду.

Найвищий рівень контролю над периферією дає програмування регістрів. Усі апаратні модулі (GPIO, USART тощо) доступні через регістри, номери яких вказані у мануалі RM0090 [19]. Однак це потребує розуміння роботи мікроконтролеру на базовому рівні. Також є базовий набір заголовкових файлів CMSIS (Cortex Microcontroller Software Interface Standard), який визначає доступ до ядра Cortex-M та дещо абстрагує доступ до регістрів [20].

Наприклад, ось як виглядає встановлення високого рівня напруги на порт PD15:

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
GPIOD->MODER &= ~(3U << (15 * 2));
GPIOD->MODER |=  (1U << (15 * 2));
GPIOD->ODR |= (1U << 15);
```

Ось що відбувається у цьому коді:

Уся периферія STM32 відключена від тактуючих сигналів за замовчуванням, тож для роботи із PD15, потрібно увімкнути генерацію тактового сигналу для порту D.

AHB1ENR (Advanced High-performance Bus 1 Enable Register) – 32-бітний регістр що відповідає за живлення периферії підключеної до шини AHB1 (GPIOA–GPIOI, DMA1, DMA2 тощо), у якому кожен біт керує одним модулем;

Далі ми встановлюємо PD15 у режим загального виходу (output push-pull).

MODER (Mode Register) – 32-бітний регістр, який визначає режим роботи кожного піну (00 – вхід, 01 – вихід, 10 – альтернативна функція, 11 – аналоговий режим). Кожен пін має два біти у цьому регістрів, наприклад, PD15 займає біти 30 та 31. Перша строка очищує біти, наступна – встановлює необхідний режим.

Остання строка встановлює високий рівень напруги на PD15.

ODR (Output Data Register) – регістр, який відповідає за напругу на пінах GPIO, які налаштовані як вихід. Операція в коді встановлює логічну “1” на цьому піні.

Очевидно, що це потужний інструмент для розробки, але це дуже складно, оскільки STM32 має загалом понад 500 регістрів, і кожен з них має десятки бітів із власним значенням та описом.

STMicroelectronics має власну бібліотеку HAL (Hardware Abstract Layer), яка має готові функції для роботи із периферією, а також має готову кодогенерацію через редактор STM32CubeIDE [21]. Ось так буде виглядати встановлення логічної “1” на PD15 із використанням HAL (однак це також потребуватиме налаштування цього піну через Pinout View у STM32CubeIDE або через *HAL_GPIO_Init*):

```
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
```

Альтернативою HAL є low-layer бібліотеки, які надають доступ до регістрів через готові макроси та структури.

Зазвичай для програмування STM32 використовують наступні середовища розробки:

- STM32CubeIDE [22] – офіційне середовище, підтримує автоматичну генерацію коду, відлагодження та емуляцію, включаючи можливість встановлювати контрольні точки та переглядати значення змінних у реальному часі;
- PlatformIO [23] + VS Code [24] – зручніше та сучасніше середовище для програмування мікроконтролерів;
- Keil uVision [25] або IAR Embedded Workbench [26] — потужні комерційні IDE.

2.4.1. Робота з периферією

Більша частина логіки вбудованих застосунків опрацьовується завдяки зовнішнім датчикам. Підбір правильних датчиків – одна із найважливіших частин проєктування електронних систем. Важливо обрати вірний інтерфейс. STM32 підтримує декілька типів інтерфейсів для комунікації.

I2C (Inter-Intergrated Circuit)

I2C (Inter-Integrated Circuit) – синхронна, одностороння послідовна шина для комунікації, винайдена у 1980-х роках [27]. Має адресацію (кожен пристрій має унікальну адресу), підтримує декілька пристроїв на одній шині. Використовується для низькошвидкісних елементів (100 кбіт/с у звичайному режимі, 10 кбіт/с у режимі заниженої швидкості, 400 кбіт/с у швидкісному режимі, до 3.4 Мбіт/с у високошвидкісному режимі; на практиці можна очікувати 100-400 кбіт/с, щоб отримати більшу швидкість потрібні короткі доріжки, сильні підтягуючі резистори – нижче 2 кОм, та підтримка високошвидкісного режиму пристроєм), найчастіше – для датчиків температури, тиску, LED/OLED-дисплеїв, гіроскопів тощо.

I2C має дві лінії, які мають бути підтягнуті до напруги живлення (зазвичай 5 В або 3.3 В): послідовна лінія даних (SDA, serial data) і лінія тактування (SCL, serial clock).

Сигнал формується за рахунок зарядженості ємності лінії, фізична ємність усієї шини обмежена 400 пФ (це включає провідники, доріжки, паразитні ємності) або 100 пФ для високошвидкісного з’єднання, оскільки при більшій ємності сигнал буде спотворений, що також обмежує кількість пристроїв.

Для адресації пристроїв зазвичай використовуються 7-бітовий адресний простір (стандарт 1992 року підтримує 10-бітну адресацію), 16 адрес зарезервовані (адреси від 0x00 до 0x0F), в тому числі:

— 0x00 – загальний виклик, для звернення до всіх пристроїв одночасно;

— 0x04 або 0x06 – Bus Reset, яка дозволяє повернути шину до стабільного стані, не є частиною стандартного набору команд, але деякі пристрої підтримують спеціальну адресу або послідовність байтів, що викликає reset;

— 0x0C – alert response, спеціальна команда, яку пристрій подає через SDA якщо є падіння напруги та яку може подати мікроконтролер, щоб з'ясувати, хто подав сигнал;

— 0x0E/0x0F – швидкісний режим, дозволяє I2C працювати до 3.4 Мбіт/с.

Передача починається із сигналу СТАРТ (SDA переходить з “1” у “0” поки SCL = “1”), за яким слідує адреса цільового пристрою та біт R/W: "0" для режиму запису, "1" для режиму читання.

Якщо пристрій існує на шині, то він відповість бітом ACK (SDA = “0”) або NACK (SDA = “1”), після чого контролер починає роботу в режимі передачі або прийому.

Перед передачею біта лінія тактування має бути у низькому рівні (SCL = 0), на лінії SDA встановлюється значення наступного біта, після чого SCL переходить у "1": це сигналізує про те, що біт готовий до зчитування. Після цього SCL повертається в "0" і можна передавати наступний біт. Дані передаються від старшого біта.

Після кожного байта передавач звільняє лінію SDA і приймач встановлює біт ACK.

Після завершення передачі даних подається сигнал СТОП (SDA з “0” переходить у “1” поки SCL = “1”).

Однією із найважливіших особливостей протоколу є розтягування тактової частоти (clock stretching), який дозволяє цільовому пристрою призупинити передачу, утримуючи SCL в низькому рівні, якщо він не готовий прийняти або надіслати наступний байт, наприклад, якщо мікросхема має повільне ПЗ та потребує часу для обробки даних).

Контролер чекає, поки SCL не переходить у "1" та чекає ще деякий час (4 мкс для 100 кбіт/с) перш ніж передати наступний біт.

У I2C є розширення SMBus, яке використовується для пристроїв з низькою пропускною здатністю. У SMBus немає довільних протоколів, лише фіксовані структури: byte, word (2 байти), block (не більше 32 байтів), process call (запис 2 байтів і миттєве читання 2 байтів), alert response (повідомлення про помилку), address resolution protocol (динамічне визначення адрес), host notify (може використовуватися ціллю для ініціації зв'язку замість контролеру), а також більш строгі таймінги, наприклад, максимальний час фронту сигналу – 1000 нс, максимальний час спаду сигналу – 300 нс.

UART (Universal Asynchronous Receiver-Transmitter)

UART – інтерфейс, який реалізує послідовну передачу даних без використання тактового сигналу, асинхронно [28]. Щоб пристрої могли спілкуватися, вони мають заздалегідь узгодити швидкість в бодах (baud rate), кількість бітів за секунду.

UART має дві лінії: TX (transmit) для передачі та RX (receive) для прийому. Вони мають підключатися хрест-навхрест, якщо на мікросхемі вони не перевернуті.

Сигнали надсилаються побітово з молодшого біта, кожен байт починається із старт-біта ("0"), бітами даних, бітом парності (parity bit) для перевірки цільності та один або два стоп-біти ("1").

Є два типи парності: парний (even parity), у цьому режимі біт парності = "1", якщо в байті непарна кількість “1”, та непарний (odd parity) – біт парності = “1” якщо в байті парна кількість “1”. Він дозволяє виявити, що під час передачі стався збій.

UART є повнодуплексним (full-duplex) інтерфейсом: передача і прийом даних здійснюються незалежно і одночасно, пристрої можуть вести двосторонню комунікацію без очікування звершення передачі в інший бік, але UART може працювати також у режимі single wire (half-duplex): передача і прийом відбувається по одній лінії (по черзі).

Також у STM32Cube IDE підтримується IrDA (Infrared Data Association): застарілий інтерфейс передачі даних через інфрачервоне світло, який

використовує інфрачервоний світлодіод і фототранзистор, інфрачервоний режим та LIN (Local Interconnect Network) – особливий протокол для автомобільної електроніки.

UART використовує передискретизацію для уточнення значення біта (over sampling): скільки разів прийомник UART перевірить значення біта ("0" чи "1"). Щоб не помилитися, UART зчитує кожен біт кілька разів (8 або 16) та обирає частіший варіант.

У стандарті також передбачене використання програмного керування (XON/XOFF), у такому випадку "старт" та "стоп" передаються як символи: 0x13 та 0x11 для зупинки та відновлення передачі.

USART (Universal Synchronous and Asynchronous Receiver-Transmitter)

USART –розширення UART. Окрім RX/TX він також має тактову лінію – SCK.

Для синхронної передачі SCK встановлюється на "0", на TX встановлюється біт, а потім SCK піднімається до "1": це сигналізує про те, що читач може прийняти біт.

Для синхронної передачі старт-бітів, як і стоп-бітів, немає, оскільки в них немає потреби.

Для USART може бути задіяне апаратне управління потоком за допомогою додаткових сигналів CTS (Clear to Send): пристрій встановлює його на "0", якщо пристрій готовий приймати і RTS (Request to Send): мікроконтролер встановлює його на "0" якщо готовий приймати.

USART також має розширення SmartCard і SmartCard with Card Clock, які призначені для роботи зі смарт-картами (SIM, банківські карти тощо) згідно стандарту ISO 7816.

SmartCard використовує USART в напівдуплексному режимі, дані передаються по одній лінії. Також задіяні специфічні налаштування кадру: 9-біт (8 для даних, 1 для парності), even parity, 1.5 або 2 стоп-біти.

Він також може мати вихід CLK (Card Clock), який подається на карту, якщо потрібен зовнішній такт та RST для скидання карти (може застосовуватися GPIO).

Також цей стандарт реалізує автоматичне управління режимом очікування, відповідями карти та таймаутами: наприклад, стандарт смарт-карт має базову одиницю часу, яка визначає тривалість одного біта.

ADC/DAC (Analog To Digial/Digital To Analog)

ADC/DAC: ADC представляє собою інтерфейс для спілкування із аналоговими датчиками.

Окрім звичайних каналів (фізичних аналогових входів) є також:

— канал для внутрішнього сенсору температури кристалу (Temperature Sensor Channel) – вбудований термосенсор для кристалі мікроконтролера, який видає аналогову напругу пропорційну температурі;

— канал внутрішньої опорної напруги (Vrefint), який видає напругу внутрішнього опорного джерела, може використовуватися для калібрування результатів ADC (особливо якщо VDD відрізняється 3.3 В);

— канал для моніторингу напруги живлення батареї (Vbat), який дозволяє вимірювати напругу на піні VBAT.

STM32 також підтримує запуск перетворення АЦП зовнішнім сигналом (external trigger for injected/regular conversion).

DAC навпаки дозволяє перетворити цифровий сигнал на аналоговий. Це використовується, наприклад, щоб вивести синусоїду або інший аналоговий сигнал.

SPI (Serial Peripheral Interface)

SPI – синхронний послідовний інтерфейс, який використовує декілька ліній: SCK для тактування, MOSI (Master Out Slave In) для передачі даних від

контролеру, MISO (Master In Slave Out) для передачу даних до контролеру, NSS (Slave Select) для вибору конкретного підлеглого.

Обмін даними відбувається побітово на кожному фронті SCK, яким керує контролер. Передача триває, допоки NSS = "0".

Дуже швидкий інтерфейс, не використовує жодних підтверджень (ACK/NACK бітів), лише таймінги. Пристрої не мають адрес, обираються через NSS. Використовуються у TFT- та OLED-дисплеях, SD-картах, для зовнішньої flash-пам'яті тощо.

CAN (Controller Area Network)

CAN – послідовний протокол передачі даних для автомобільної електроніки, реалізує передачу повідомлень для усіх приймачів у мережі (усі вузли бачать усі повідомлення).

FSMC (Flexible Static Memory Controller)

FSMC – інтерфейс для контролерів пам'яті, що дозволяє підключати зовнішні пристрої пам'яті або будь-яку іншу периферію таким чином, ніби вони є частиною внутрішнього адресного простору.

При підключенні датчика через FSMC, мікроконтролер адресує зовнішній пристрій як звичайну змінну у пам’яті. FSMC працює на апаратному рівні, без участі CPU.

Тоді, наприклад, код

```
*(volatile uint16_t *)0x60000000 = …;
```

запише значення прямо у датчик або SRAM.

SDIO (Secure Digital Input Output)

SDIO – інтерфейс для швидкісної роботи з SD-картами, що дозволяє обмінюватися командами SD-протоколу через пін CMD.

DCMI (Double Crossover Merging Interchange)

DCMI – інтерфейс, що може приймати зображення від CMOS-камер у форматах YUV, JPEG тощо.

I2S (Inter-IC Sound)

I2S – послідовний інтерфейс для аудіо.

RNG (Random Number Generator)

RNG – інтерфейс для генерації випадкових чисел, який працює на джерелі відхилень або на шумі RC-ланцюга.

CRC (Cyclic Redundancy Check)

CRC – інтерфейс для обчислення контрольної суми CRC, яка може бути використана для перевірки цілісності даних, працює апаратно.

STM32 також може працювати з периферією різними способами.

Polling Mode

Режим опитування – режим, у якому програма у циклі перевіряє, чи готовий пристрій (постійно читає статусний регістр). Найпростіший із режимів роботи, але неефективний, бо процесор постійно занятий.

```
while (!(USART1->SR & USART_SR_TXE));
USART1->DR = 'A';
```

У цьому прикладі ми читаємо статусний регістр SR модуля UART1. Біт TXE (Transmit Data Register Empty) означає, що передавальний регістр порожній. Коли біт стане порожнім, передаємо у DR (data register) дані (символ "A" у цьому випадку).

Blocking Mode

Блокуючий режим – обгортка HAL бібліотеки для режиму опитування.

```
char data = 'A';
HAL_UART_Transmit(&huart1, (uint8_t*)&data, 1, HAL_MAX_DELAY);
```

У цьому прикладі ми передаємо дані, використовуючи *HAL_UART_Transmit*, який працює у блокуючому режимі, “1” – кількість байтів, “HAL_MAX_DELAY” – час очікування (процесор буде заблокований до завершення).

Interrupt Mode

При роботі у режимі переривання, сама периферія подає сигнал у NVIC, який викликає обробник, коли щось відбулось. Дозволяє іншим задачам працювати паралельно.

Щоб запустити прийом одного байту через переривання, потрібно використати функцію *HAL_UART_Receive_IT*, а також додати функцію-обробник *HAL_UART_RxCpltCallback*:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == USART1)
  {
      // …
  }
}
```

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було розглянут**о** основні етапи створення електронних систем, включно із особливостями вибору компонентів, фізичними особливостями проєктування принципових електричних схем та друкованих плат, типами монтажу, властивостями матеріалів.

Було переглянуто ключові елементи електричних схем: резисторів, конденсаторів, діодів, MOFSET-транзисторів, а також приділено увагу особливостям живлення мікроконтролеру STM32, стабілізації напруги, вимогам до струмового навантаження, вибору ширини доріжок та захисту від наведень та імпендансного узгодження.

Особливу увагу було приділено схемі захисту живлення — як одному з критично важливих елементів пристрою, що забезпечує стійкість до нестабільностей джерела, захист від перенапруги, переполюсування, перевищення струмового навантаження та збоїв у ланцюзі живлення. Було також розглянуто практичні варіанти реалізації захисту за допомогою стабілізаторів, діодів, конденсаторів, запобіжників та фільтрів.

Також окремо розглянуто методи виробництва друкованих плат: структуру шарів, фінішне покриття, варіанти монтажу компонентів: наскрізного і поверхневого, їх переваги та обмеження.

Викладених в розділі принципів та особливостей достатньо для необхідної бази для подальшої реалізації навчально-методичного комплексу, який би дозволив студентам не тільки повторити готові приклади, але й усвідомити логіку проєктних рішень.

РОЗДІЛ 3. РОЗРОБКА АВТОНОМНОЇ МЕТЕОСТАНЦІЇ НА БАЗІ STM32

3.1. Аналіз вимог та вибір датчиків

Для підсилення практичної цінності матеріалу, було розроблено власний пристрій на базі мікроконтролеру STM32, а саме автономну метеостанцію.

Проведемо попередній аналіз вимог до системи:

- метеостанція має забезпечувати щонайменше базову інформацію про навколишнє середовище, якої було б достатньо для опису та прогнозування погоди: тиск, температура, вологість, стан погоди (зокрема, рівень хмарності, наявність опадів тощо): цих даних достатньо, наприклад, для індексу Замбретті (90% точності для 12-годинних прогнозів) або машинного навчання чи прогнозу на основі трендів;
- розроблюваний прилад має працювати автономно: передбачається його розміщення у віддалених локаціях, тож він має бути енергоефективним і мати можливість зарядки через USB;
- дані, що були зібрані з навколишнього середовища, мають передаватися на віддалений веб-сервер для їх подальшого аналізу через мережу Інтернет за допомогою стільникового модему;
- програмне забезпечення має підтримувати отримання команд від веб-серверу та їх виконання;
- прилад має підтримувати інтерфейс відладки SWD (Serial Wire Debug) для оновлення прошивки за допомогою програматору накшталт ST-Link 2.0 [29];
- прилад має бути стійким до вологи та температури.

Для реалізації вищеописаного функціоналу було обрано набір датчиків для збору даних із навколишнього середовища.

Основними критеріями були:

— стабільність і точність вимірювань;
— стійкість до вологості та температури;
— низьке енергоспоживання та електромагнітна сумісність;
— апаратна сумісність із мікроконтролером STM32;
— наявність даташитів, за можливості, готових бібліотек та прикладів (для будь-якої з платформ);
— компактність та простота монтажу на плату.

Виконавши аналіз, було обрано підходящі датчики.

BME280

BME280 – комбінований датчик, що забезпечує одночасне вимірювання температури, вологості та тиску з високим ступенем роздільності (датчик здатен визначити навіть невелику зміну висоти (1.7 см) та має незначну похибку вимірювання вологості (3 %)).

Цей датчик поєднує в одному корпусі температурний сенсор побудований на основі кремнієвого терморезистора, опір якого змінюється залежно від температури та перетворюється в цифрове значення через вбудований аналогово-цифровий перетворювач; ємнісний сенсор вологості: між двома електродами розміщений гігроскопічний полімер, що, поглинаючи вологу з повітря, змінює власну провідність, що призводить до зміни ємності; п'єзорезистивний елемент, розміщений на кремнієвій мембрані, який деформується під впливом атмосферного тиску;

Його було обрано оскільки використання лише одного датчику замість трьох дозволяє суттєво зменшити розміри пристрою (розміри самого датчику: 2.5 мм × 2.5 мм × 0.93 мм), на відміну від свої альтернатив. Наприклад, DHT11 вимірює лише температуру та вологість при меншій точності та вищому енергоспоживанні, а BME280 – старше покоління датчиків Bosch, не підтримує вимірювання вологості;

Цей датчик працює у широкому діапазоні температур (від -40 °C до +85 °C) та тиску: від 300 до 1100 гПа (цей діапазон охоплює усі можливі атмосферні тиски від рівня моря до гірських умов);

Він також має низьке енергоспоживання: 3.6 мкА у режимі роботи, 0.1 мкА у режимі сну та широкий діапазон робочої напруги: від 1.8 В до 3.6 В, а деякі окремі специфікації підтримують до 5В.

BH1750

BH1750 для вимірювання освітленості, який використовує фотодіод, який генерує струм пропорційно інтенсивності падаючого світла. Цей струм накопичується в інтегруючому конденсаторі протягом певного часу, а потім це значення оцифровується внутрішнім аналогово-цифровим перетворювачем.

Цей датчик було обрано оскільки його спектральна чутливість відповідає людському оку (560 нм, пік зелено-жовтої області спектра), він не потребує додаткової калібровки (окрім як використання корекційного коефіцієнту у окремих випадках, наприклад, якщо датчик розміщено за склом), споживає 1 мкА у сплячому режимі та високу чутливість: до 1 люксу.

VEML6070

Датчик ультрафіолетового випромінювання VEML6070. Цей датчик працює аналогічно до датчику освітленості, але фотодіод сприймає світло довжиною приблизно 320-280 нм.

Цей датчик дозволяє визначити наявність хмар, оскільки вони частково або повністю блокують ультрафіолетове випромінювання. Одночасне застосування BH1750 та VEML6070 дозволяє оцінити ступінь прозорості атмосфери (хмари, дим, туман тощо).

Цей датчик було обрано, оскільки він, на відміну від багатьох аналогів, наприклад, SI1145, дійсно вимірює інтенсивність ультрафіолетового випромінювання, а не віртуальний індекс, не потребує додаткової калібрації (окрім як для стандартизації UV-індексу), є енергоефективним: споживає 100 мкА під час роботи та 1 мкА у сплячому режим та має дуже просту інтеграцію із мікроконтролером.

FC-37

Аналоговий датчик FC-37 для вимірювання наявності опадів. Представляє собою дощову пластину з відкритими провідниками та невелику мікросхему із підсилювачем та компаратором сигналу, які і формують аналоговий і цифровий виходи.

Працює наступним чином: коли на пластину потрапляє вода, між доріжками виникає провідність, яка змінює аналоговий сигнал, що дозволяє визначити факт дощу і приблизно оцінити інтенсивність опадів.

Його було обрано через простоту підключення та простий механізм, який можна доволі просто перенести на плату для мініатюризації.

Усі інші аналоги, хоча й є більш точними (наприклад, дають кількісну оцінку мм/год) та довговічними (пластина окислюється), однак є дорожчими та складнішими, зазвичай вони представляють собою датчики типу “tipping bucket” (його також називають ковшовим опадомір: всередині сенсора розташовано дві чашки, які гойдаються під дією ваги, що закриває контакт та генерує електричний імпульс) або оптичні сенсори.

SIM7600E-H LTE Cat-4

Модуль зв'язку SIM7600E-H LTE Cat-4 – стільниковий модем для комунікації із зовнішнім світом. Цей модуль працює за допомогою AT-команд (типовий набір текстових команд для модулів зв'язку) через UART інтерфейс. Його було обрано оскільки він підтримує високошвидкісний 4G-зв’язок, має вбудований тримач для SIM-карти та SD-карти та підтримує протоколи TCP/IP, HTTP, MQQT.

Варто зазначити, що він має досить високе енергоспоживання та доволі складно програмується, однак єдиними аналогами є Wi-Fi модулі накшталт ESP32 (тоді як наш пристрій має працювати у польових умовах), LoRa модулями (бездротова передача даних за допомогою радіозв'язку на відстань 10-15 км) або компактніші, але застарілі модулі типу SIM800L, які підтримують лише 2G/3G.

PCA9546A

Мультиплексор PCA9546A дозволяє підключати одночасно до 4 пристроїв I2C. Хоча до самої шини можна підключити більше сотні пристроїв, на усіх модулях припаяні резистори, які будуть тягнути SDA/SCL лінії, тож використання мультиплексора ізолює резистори один від одного.

3.2. Алгоритм роботи мікроконтролеру

Для роботи пристрою достатньо збирати дані з певним інтервалом часу, а оскільки однією із основних вимог до пристрою є його енергоефективність, ми будемо використовувати STOP-режим мікроконтролеру та вмикати пристрій за допомогою RTC, який буде генерувати wake up переривання кожні N хвилин (функцію *HAL_RTCEx_WakeUpTimerEventCallback*).

Основний цикл роботи мікроконтролеру виглядає наступним чином:

```
while (1)
{
  if (!sleeping)
  {
    sleeping = true;
    // ...
 }

  __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);

  HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,
PWR_STOPENTRY_WFI);

  SystemClock_Config();

  HAL_I2C_DeInit(&hi2c2);
  __HAL_RCC_I2C2_CLK_ENABLE();
  MX_I2C2_Init();

  HAL_UART_DeInit(&huart2);
  __HAL_RCC_USART2_CLK_ENABLE();
  MX_USART2_UART_Init();

  HAL_Delay(50);
}
```

Для виходу із режиму сна ми будемо використовувати булеву змінну, оскільки у обробниках переривань критично важливо виконувати мінімальну кількість дій, щоб запобігти переповненню стеку, вкладеним перериванням та непередбачуваної поведінки системи в цілому. В нашому випадку ми будемо змінювати значення булевої змінні *sleeping*.

__HAL_PWR_CLEAR_FLAG представляє собою макрос, який очищує прапорець пробудження. Це потрібно, щоб не сталося негайного повторного пробудження.

HAL_PWR_EnterSTOPMode зупиняє роботу програми. Після переривання програма продовжить працювати з цього місця.

У режимі STOP тактова генерація зупиняється або змінює режим роботи, тому нам потрібно відновити нормальну роботу системного годинника та деініціалізувати I2C, щоб налаштувати периферію: для цього потрібно повторно увімкнути її тактування та конфігурувати.

Структурна та функціональна схема (діаграма класів) наведені у Додатках Б та В. Весь код наведений у Додатку Ґ.

3.3. Проєктування електричної принципової схеми

Електрична принципова схема пристрою складається із декількох модулів: схема живлення і зарядки, схема мікроконтролеру, схема стільникового модему, схема I2C мультиплексору, схема датчиків.

Окремо розглянемо схему живлення (рис. 3.1) та схему FC-37 (рис. 3.2).

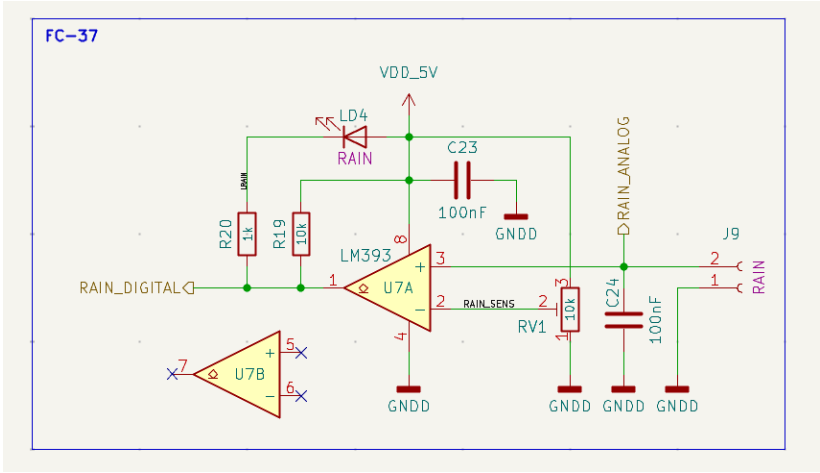


Рисунок 3.1 – Символ FC-37

Розглянувши фізичний модуль FC-37 та дослідивши механізм його роботи, було встановлено що аналоговий сигнал порівнюється компаратором LM393 з опорним значенням напруги, який регулюється за допомогою потенціометра.

Якщо рівень напруги перевищує опорне значення, компаратор формує цифрову “1”. Окрім того, аналоговий вихід дозволяє отримати дані напряму з пластини.

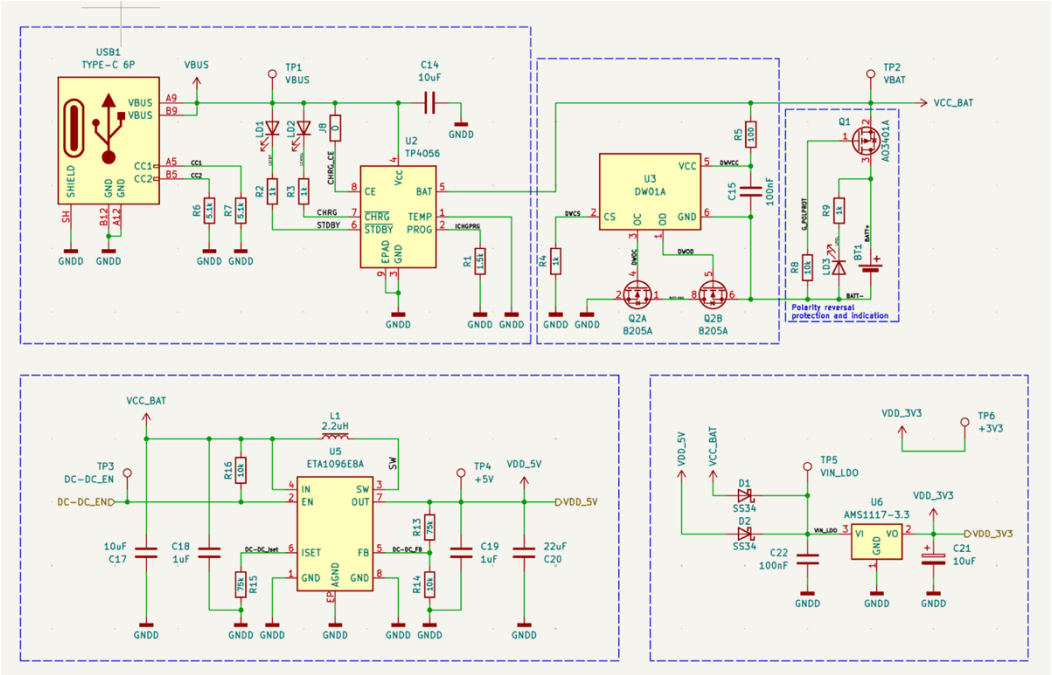


Рисунок 3.2 – Принципова схема живлення

Оскільки пристрій має бути автономним, схема живлення представляє собою комплексну систему із зарядним пристроєм та перетворювачами напруги.

Мікросхема TP4056 керує процесом зарядки, для захисту акумулятору використовується мікросхема DW01A із двома польовими транзисторами типу 8205A для захисту від перезаряду, надмірного розряду та короткого замикання. Ці транзистори працюють подібно до електричних ключів: при напрузі нижче 2.5 В або вище 4.3 В, вони розмикають коло.

Щоб отримати стабільні 5 В використовується мікросхема ETA1096E8A, а 3.3 В – AMS1117-3.3.

Окремо варто зазначити, що TX/RX вже схрещені на модулі SIM7800E, а підключення відбувається через гребінку 2x20 (на платі знаходиться конектор для Raspberry Pi).

Мікроконтролер використовує наступні піни:

- PB10/PB11 (I2C2) для підключення мультиплексору;
- PD8/PD9 (USART2) для підключення модему, PE0 (GPIO Output) використовується для контролю живлення модеми;
- PA1 (ADC1_IN1) для аналогового виходу датчику опадів FC-37;
- PA13/PA14 використовуються як виходи SWDIO/SWCLK для інтерфейсу SWD.

Принципова схема наведена у Додатку А,

3.4. Проєктування друкованої плати

Для пристрою була спроєктована чотирьохшарова друкована плата.

На верхньому шарі (F.Cu) був розміщений тримач для батареї BH-18650-A5BJ001-2D, дворядний піновий роз'єм для стільникового модему та однорядний – для SWD-порту. Також на ньому проведено доріжку для живлення 5 В, а решта шару залито полігоном землі.

Перший внутрішній шар (In1.Cu) є суцільним полігоном землі.

На другому внутрішньому шарі (In2.Cu) проведено доріжку для живлення 3.3 В, яке необхідно для мікроконтролеру.

Датчики були розміщені на нижньому шарі (B.Cu). Для підключення пластини FC-37 був використаний конектор типу XH-2A. Для інших датчиків було використано пінові роз'єми із кроком 2.54 мм (PM2.54-1*4, PM2.54-1*5) під пайку. Решта нижнього шару залито полігоном землі.

Друкована плата наведена у Додатку Г.

3.5. Тестування розробленого пристрою

Зібрана плата виглядає наступним чином (рис. 3.3, 3.4):

Рисунок 3.3 – Верхній шар плати

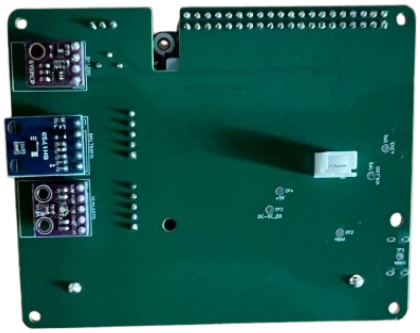


Рисунок 3.4 – Нижній шар плати

Для тестування датчиків ми можемо перевірити дані з них прямо у режимі відладки у STM32Cube IDE (інтерфейс SWD дозволяє відладку у режимі реального часу, а STM32Cube IDE дозволяє переглядати значення змінних).

Для відладки було використано наступний код:

```
float illuminance;
uint32_t intensity;
float t, h, p;
uint32_t wetness;
```

```
BH1750_Read(&illuminance);
HAL_Delay(150);

VEML6070_Read(&intensity);
HAL_Delay(150);

FC37_Read(&surface_wetness);
HAL_Delay(150);

BME280_Init(&config);
BME280_Read(&t, &h, &p);
```

BME280_Init виконує функцію калібрування датчику BME280: цей датчик містить коефіцієнти у внутрішніх регістрах, які записані виробником під час заводського калібрування.

Результат виконання можна побачити на рис. 3.5.

Name	Type	Value
illuminance	float	178.333328
intensity	uint32_t	2
t	float	28.6499996
h	float	39.4453125
p	float	997.893738
wetness	uint32_t	4095

Рисунок 3.5 – Результат зчитування даних з датчиків у приміщенні

Під час тестування пластина FC-37 була повністю суха, датчики освітленості знаходилися у декількох метрів напроти вікна. Вимірювання робилося у приміщенні 23 травня 2025 року, температура повітря згідно з онлайн-сервісами – 23-24 градуси, вологість – 51 %.

Спробуємо помістити пластину у воду, а на датчики освітленості посвітимо ліхтариком. Результати зчитування можна побачити на рис. 3.6.

Name	Type	Value
illuminance	float	4888.33301
intensity	uint32_t	5
t	float	27.9899998
h	float	39.0214844
p	float	997.872314
wetness	uint32_t	86

Рисунок 3.6 – Результат зчитування даних з датчиків під зовнішнім впливом

Як бачимо, значення освітленості та інтенсивності ультрафіолетового випромінювання збільшилися, як і показник вологості (FC-37 повертає значення від 0 до 2^12-1, де 0 – найбільше значення вологи).

На основі цих вимірювань можемо зробити висновок, що датчики підключені та запрограмовані вірно.

З метою подальшого використання даних з датчиків, було написано веб-сервер на TypeScript [30] із використанням Node [31] та express.js [32], який приймає та зберігає дані у форматі CSV.

TypeScript представляє собою мову, яка розширює JavaScript [33] статичною типізацією, поєднуючи гнучкість JavaScript із безпекою типів, надійністю та масштабованістю. Node представляє собою найпопулярніше середовище для запуску JavaScript на сервері, а express.js є найпростішим і водночас гнучким фреймворком для HTTP-серверів.

Код серверу наведений у Додатку Ґ. Результат прийому команд зображений на рис. 3.7.

```
anton@Antons-MacBook-Pro server % node .
[2025-05-24T16:10:42.955Z] GET /?l=6.67&i=0&t=27.19&h=42.16&p=999.32&w=4095
[2025-05-24T16:11:20.146Z] GET /?l=6.67&i=0&t=27.14&h=42.21&p=999.24&w=4095
[2025-05-24T16:11:57.160Z] GET /?l=0.00&i=0&t=27.12&h=42.79&p=999.37&w=4095
[2025-05-24T16:12:34.146Z] GET /?l=6.67&i=0&t=26.95&h=42.83&p=999.38&w=4095
[2025-05-24T16:13:11.189Z] GET /?l=6.67&i=0&t=26.90&h=42.70&p=999.36&w=4095
[2025-05-24T16:13:48.364Z] GET /?l=6.67&i=0&t=26.89&h=42.74&p=999.29&w=4095
```

Рисунок 3.7 – Результат прийому даних

Було зібрано та проаналізовано дані з 24 травня 23:00 по 25 травня 03:30 з використанням бібліотеки Matplotlib [34] (Python [35]). Цей вибір обґрунтований тим, що ці технологій є де-факто стандартом для роботи з даними, оскільки мають широкі можливості з візуалізації та аналізу даних. Код відображення даних наведений у Додатку Ґ. Отримані значення були візуалізовано та порівняно з даними з онлайн-сервісів прогнозу погоди (рис. 3.8).

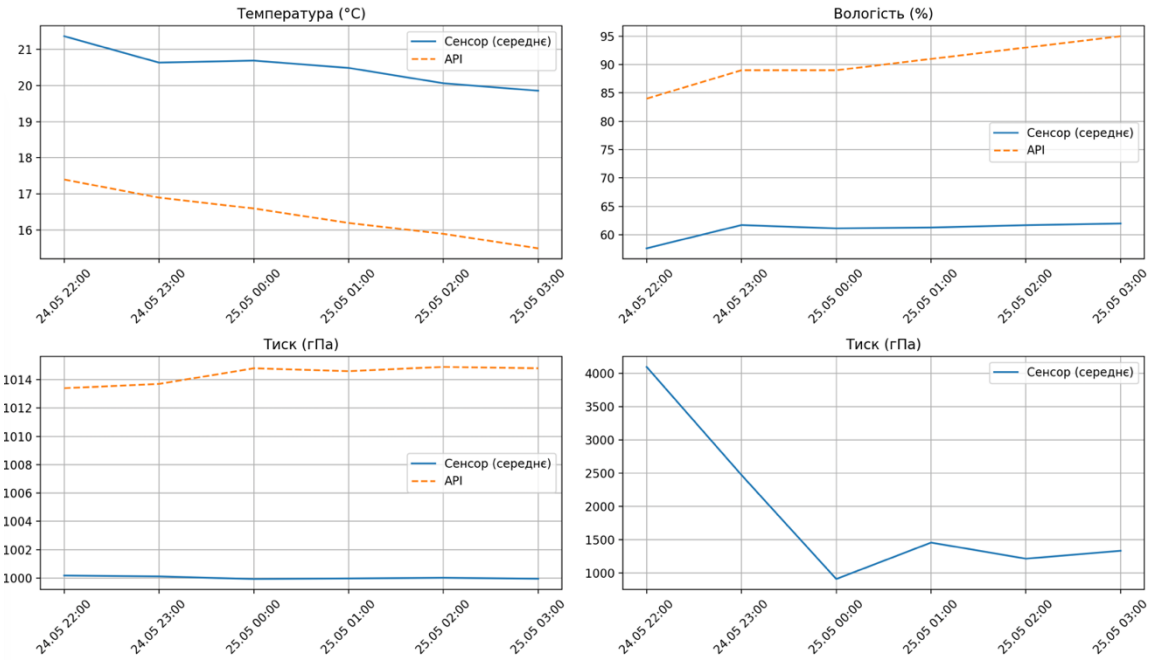


Рисунок 3.8 – Відображення зібраних даних

На веб-ресурсі дані зберігаються погодинно, тож дані з сенсорів були теж згруповані погодинно.

З отриманих даних можна зробити наступні висновки, що протягом дослідження відбулося помітне підвищення вологості, що супроводжувалося коливаннями тиску та короткочасним зниженням температури, що вказує на утворення опадів.

Бачимо доволі суттєву похибку, тож обчислимо її. Результат обчислень можна побачити на рис. 3.9.

```
Температура: RMSE = 4.10 (24.98 %)
Вологість: RMSE = 29.35 (32.56 %)
Тиск: RMSE = 14.36 (1.42 %)
```

Рисунок 3.9 – Середньоквадратична похибка

Оскільки вимірювання було зроблено в приміщенні, очікувана різниця до 5 градусів Цельсія, до 20 % вологості та зовсім невелике зміщення тиску. Як бачимо, результати цілком співпадають, різницю можна пояснити безпосередньо появою опадів та неточністю вимірів (оскільки дані згруповані). На основі цих вимірювань можемо зробити висновок, що ПЗ реалізовано вірно.

ВИСНОВОК ДО РОЗДІЛУ 3

Було розроблено автономну метеостанцію на базі мікроконтролеру STM32, на основі якої буде ґрунтуватися навчально-методичний комплекс для проєктування електронних систем.

У процесі роботи були детально проаналізовані вимоги до пристрою та обрані цифрові сенсори для вимірювання температури, вологості, тиску, освітленості, інтенсивності ультрафіолетового випромінення та наявності опадів, а також стільниковий модем для комунікації із веб-сервером та зберіганні даних на SD-карті.

Також розроблено алгоритм мікроконтролеру з урахуванням вимог до енергоефективності пристрою: з використанням STOP-режиму та періодичного пробудження через переривання RTC.

На основі функціональної схеми та вимог до живлення, було створено електронну принципову схему пристрою та спроєктовано друковану плату.

Як результат, було отримано та налаштовано готовий пристрій, а також розроблено програму для збору та відправки даних з навколишнього середовища на сервер.

Дані з датчиків були передані на сервер, де були збережені у форматі CSV. Було виконано побудову графіків за допомогою Python та бібліотеки Matplotlib. Дані, отримані з датчиків, були співставленні з даними онлайн-сервісів для прогнозу погоду. Було також розраховано середньоквадратичну похибку.

Отримане відхилення знаходяться у допустимих межах, особливо враховуючи, що сенсори були розміщені у приміщенні, що підтверджує правильність роботи пристрою.

РОЗДІЛ 4. РОЗРОБКА НАВЧАЛЬНО-МЕТОДИЧНОГО КОМПЛЕКСУ ДЛЯ ПРОЄКТУВАННЯ ЕЛЕКТРОННИХ СИСТЕМ

Оскільки темою роботи є створення навчально-методичного комплексу для проєктування електронних систем, доцільно пояснити студентам, як користуватися інтерфейсом KiCad 9 для створення електричних схем та друкованих плат.

Відкривши KiCad, бачимо вікно керування проєктом (рис. 4.1). Змінити мову на українську можна через меню *Preferences* > *Set Language*.

Натискаємо Новий проєкт, вводимо назву та зберігаємо. Після цього у списку *Файли проєкту* мають з'явитися файли із розширенням PCB (друкована плата) та SCH (електрична схема).

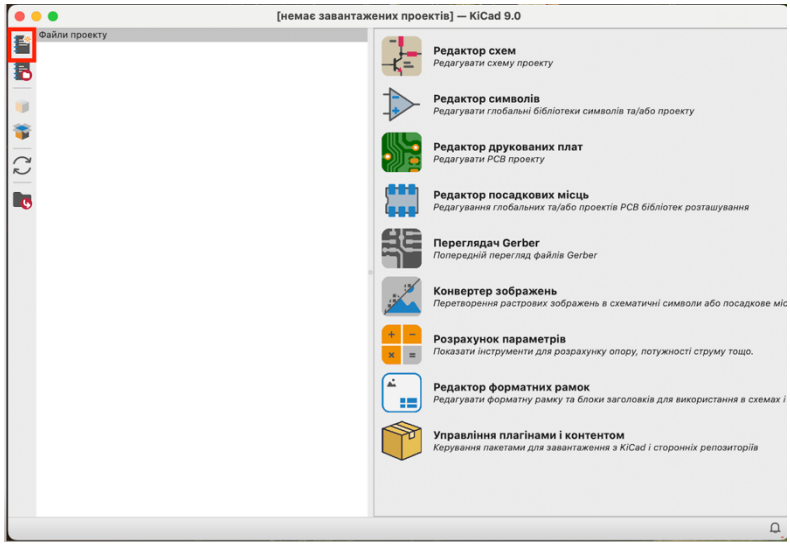


Рисунок 4.1 – Вікно керування проєктом

Натиснувши двічі на файл із розширенням SCH, має відкритися редактор електричних схем (рис. 4.2).

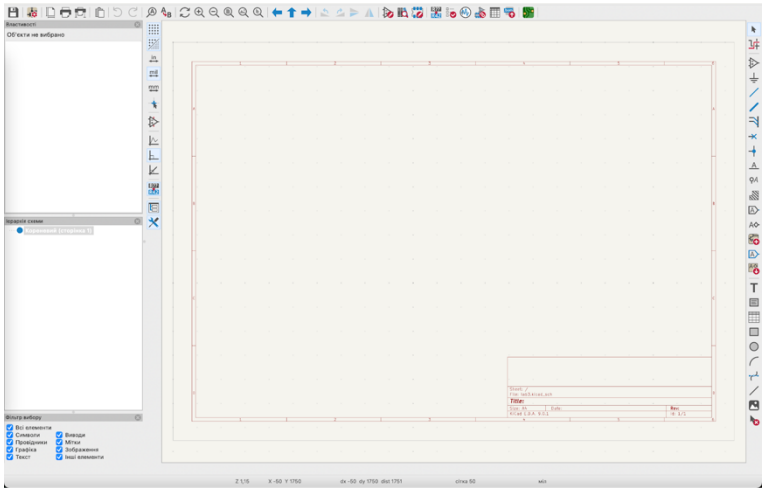


Рисунок 4.2 – Редактор електричних схем

Розглянемо його детальніше. Зліва бачимо вікна властивостей, де відображаються властивості вибраного елемента (компонента, мітки тощо) та ієрархія схеми, де відображена структура проєкту (схема може складатися з декількох сторінок).

Посередині – робоча зона та штамп. Натиснувши на штамп, відкриється вікно Параметрів сторінки (відкрити можна через меню *Файл* > *Налаштування сторінки*), де можна вказати назву схеми, автора тощо.

З усіх боків бачимо панелі інструментів. Розглянемо деякі (рис. 4.3).

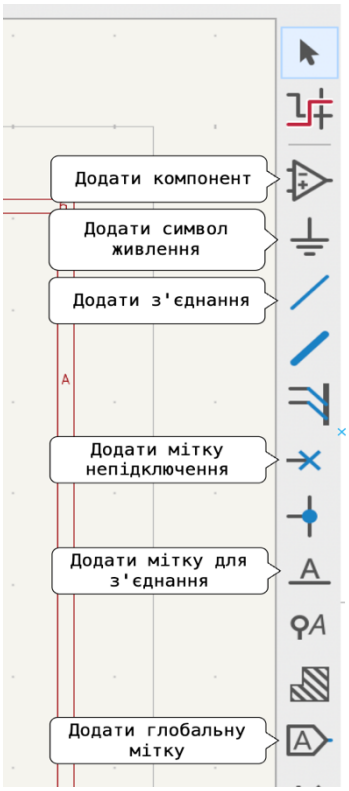


Рисунок 4.3 – Права панель інструментів у середовищі KiCad

Додати компонент відкриває бібліотеку компонентів. KiCad має тисячі готових компонентів (резисторів, мікроконтролерів, конденсаторів тощо).

Додати символ живлення дозволяє додати спеціальний символ живлення на схему (VDD/GND), які показують підключення до джерела живлення або землі.

Додати з'єднання використовується для малювання ліній, що логічно з'єднують виводи компонентів у схемі. Якщо лінія не під'єднана до виходу компоненту, потрібно клікнути двічі, щоб її завершити.

Додати мітку непідключення дозволяє додати прапорець, який вказує, що певний вивід компонента свідомо залишений без підключення.

Додати мітку для з'єднання дозволяє додати текстову мітку до провідника, що дозволяє ідентифікувати сигнали та з'єднувати точки без прямих ліній між ними.

Додати глобальну мітку дозволяє додати мітку, яка поширюється на всю схему — сигнал із цією міткою автоматично підключений до всіх інших з такою ж назвою, незалежно від сторінок чи блоків.

Додати ієрархічну мітку дозволяє додати мітку, яка може передати сигнали між основною схемою та підсхемами.

У верхній панелі інструментів нас також цікавить *Перевірка електричної схеми* (рис. 4.4), яка здатна знайти у схемі електричні помилки накшталт нез'єднаних виводів або конфліктів між типами сигналів.

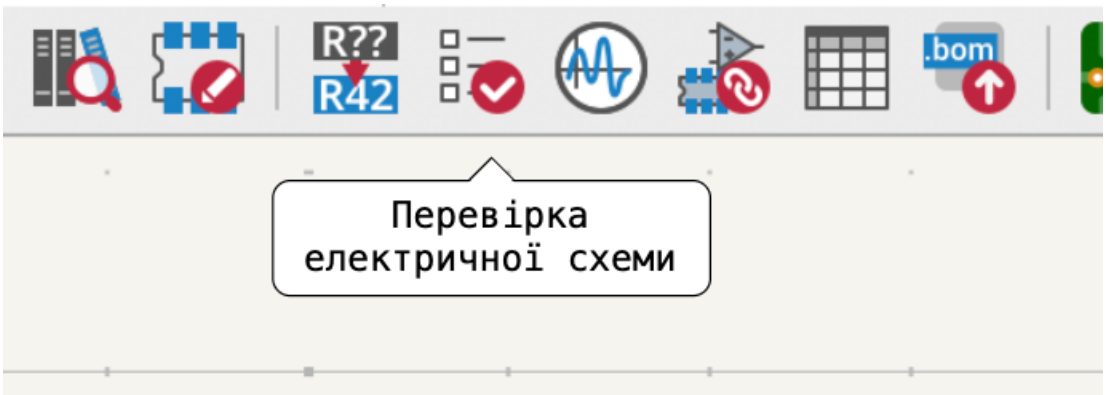


Рисунок 4.4 – Верхня панель інструментів у середовищі KiCad

Щоб додати компонент до схеми, натиснемо на *Додати компонент*. Вікно вибору компонентів можна побачити на рис. 4.5.

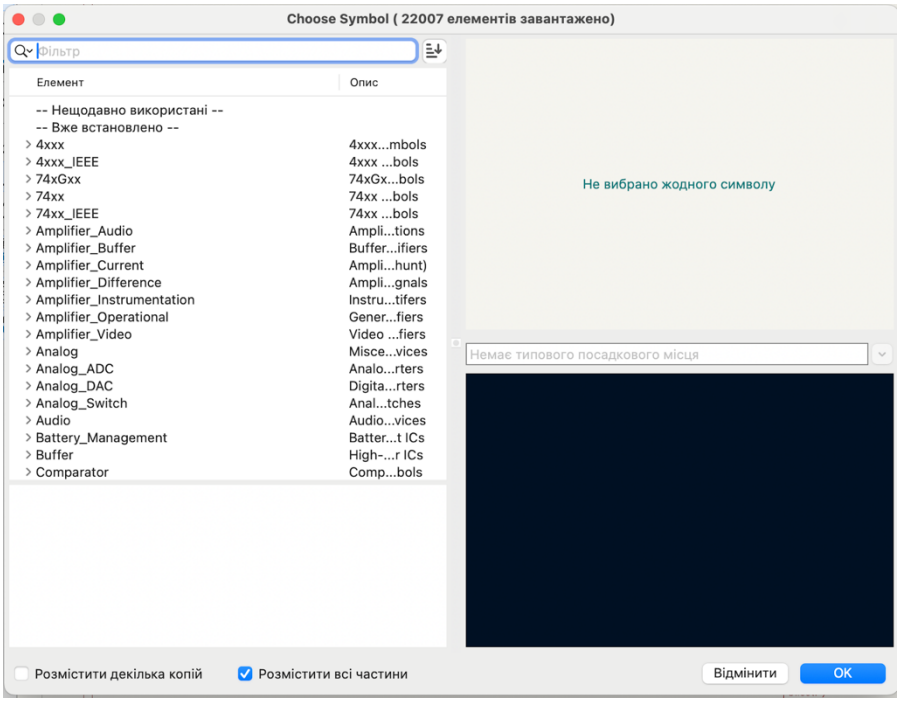


Рисунок 4.5 – Вікно вибору компонентів

У відкритому вікні бачимо *дерево бібліотеки компонентів*. KiCad має тисячі готових символів для використання у електричних схемах. Справа розташовані *перегляд символу* та *перегляд посадкового місця* (як компонент буде виглядати на платі).

Додамо на схему резистор (кодова назва у бібліотеці: *R*). Натиснувши X, можна віддзеркалити символ горизонтально, а натиснувши R, здійснити поворот на 90 градусів.

Двічі натиснувши на символ, відкриється *вікно властивостей компоненту* (рис. 4.6).

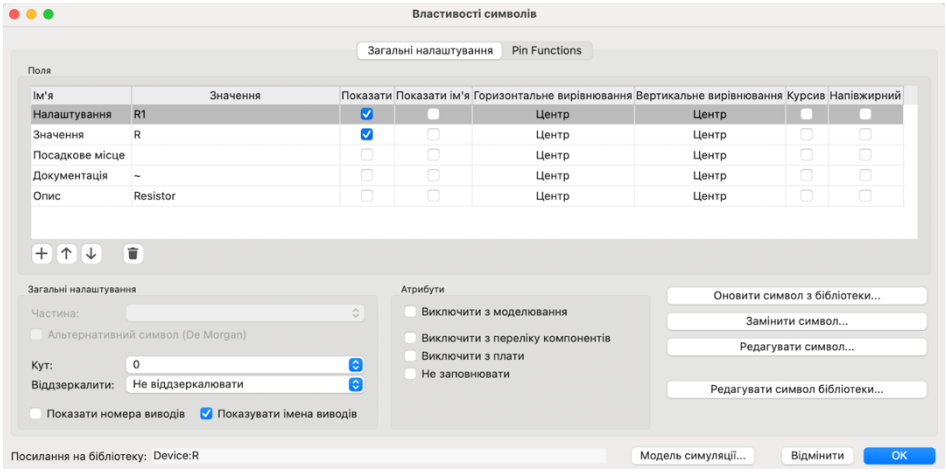


Рисунок 4.6 – Вікно властивостей компонентів

Тут можна встановити *значення* (номінал) компоненту (опір, ємність тощо). Прийнято використовувати літературні позначення: 10 – 10 Ом, 10k - 10 кОм тощо.

Додамо аналогічним чином на схему батарейку (кодова назва *Battery_Cell*).

Тепер поєднаємо компоненти (рис. 4.7). Для цього ми можемо просто з'єднати кінці проводів мишкою. Для цього наводимо курсор на вивід компоненту, нажимаємо ЛКМ, ведемо лінію до виводу іншого компоненту та нажимаємо ЛКМ на кінцевій точці. Для створення кутових поворотів треба просто змінити напрям миші. Щоб скасувати поточне з'єднання, можна нажати Esc. Щоб скасувати створене з'єднання – Ctrl+Z.

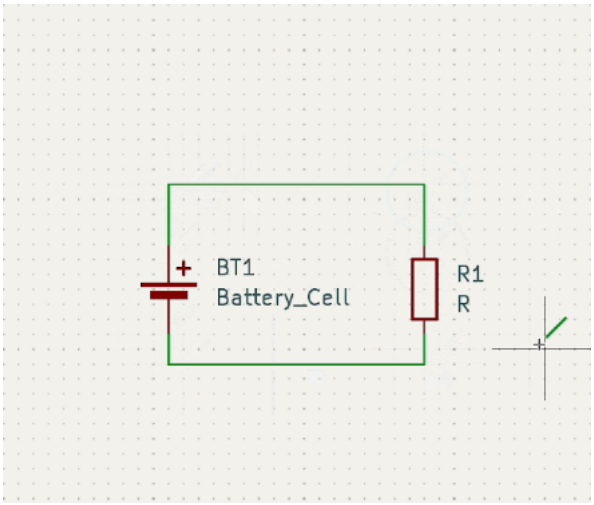


Рисунок 4.7 – Підключені компоненти у середовищі KiCad

Тепер присвоїмо компонентам посадкові місця. Для цього в меню оберемо *Інструменти* > *Редагувати поле символу* (рис. 4.8).

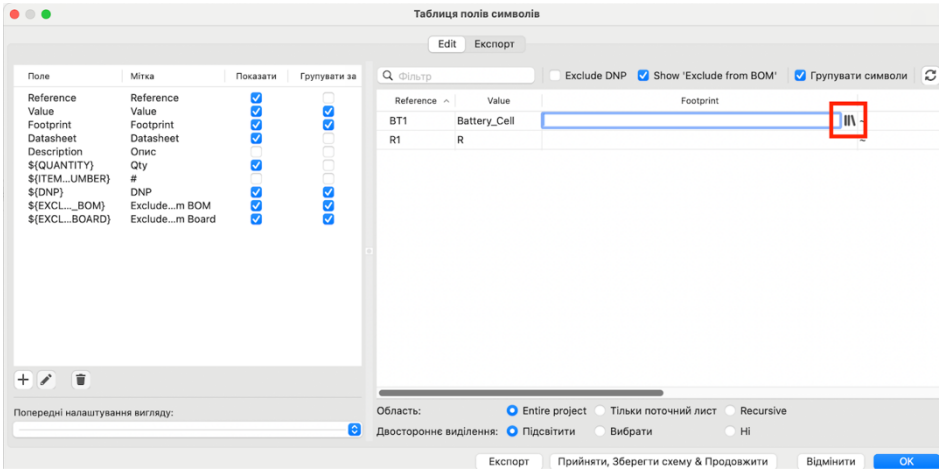


Рисунок 4.8 – Редактор полів символів

Натиснувши на поле посадкового місця, з’явиться іконка, натиснувши на яку, відкриється вікно вибору посадкового місця (рис. 4.9).

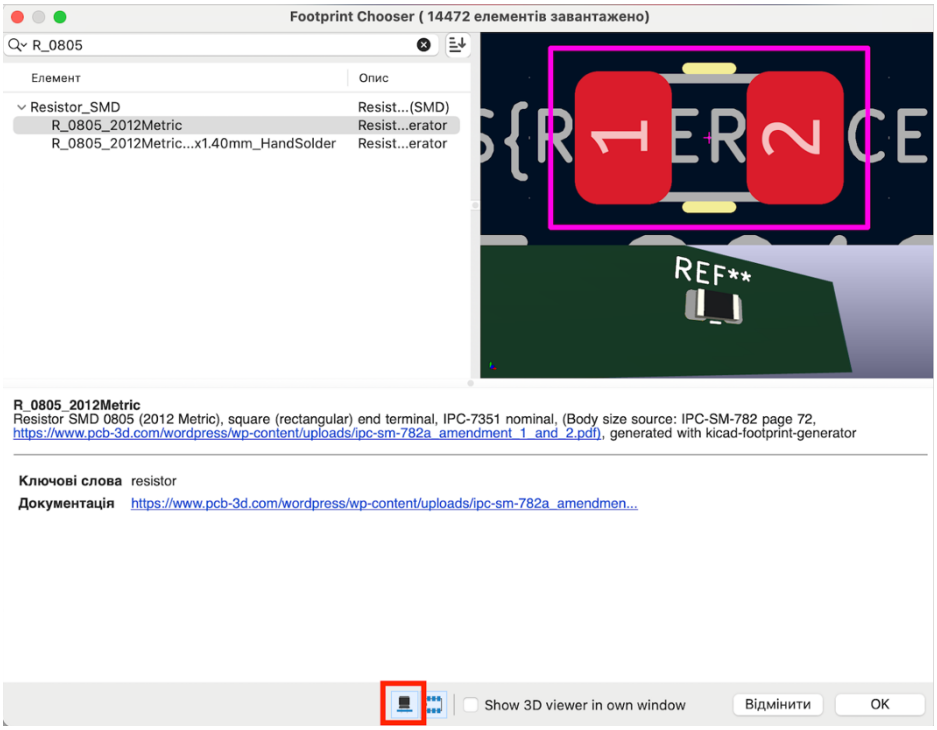


Рисунок 4.9 – Вибір посадкового місця

Звісно, призначити посадкове місце можна прямо при виборі компонента із бібліотеки, але тут можна одразу переглянути, як компонент буде виглядати на платі: для цього потрібно увімкнути переглядач іконкою внизу вікна.

Назви компонентів можуть здаватися дещо заплутаними, але вони містять усю необхідну інформацію. Вони складається з типу компонента (наприклад, R для резистора), його розміру у дюймах (наприклад, 0805) та іноді метричного еквівалента (наприклад, 2012Metric).

Наприклад, *R_0805_2021Metric* у категорії Resistor_SMD означає, що це SMD-компонент розміром 0.08 x 0.05 дюймів або 2.0 x 1.2 мм.

Перед тим, як працювати із мікроконтролером, у властивостях схеми (*Файл > Властивості схеми*) у вкладці *Рівні порушення* для правила *Вивід не підключений* встановимо значення *Ігнорувати* (це дозволить нам не встановлювати маркери непідключення на усі невикористовувані піни мікроконтролеру).

Для зручності поки виведемо VDD та GND із батареї.

Додатково для батареї змінимо функції виходів. Щоб переглянути встановлені функції, треба обрати вкладку *Функції виводів* (рис. 4.10).

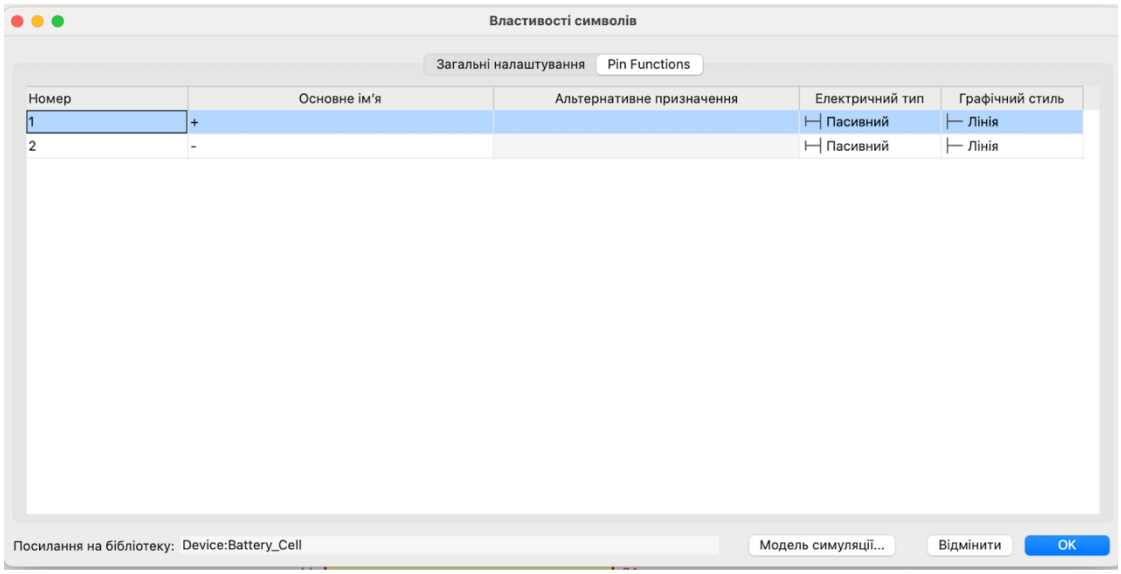


Рисунок 4.10 – Вибір функції виводів

Щоб змінити функцію, треба перейти у вкладку *Загальні налаштування* та натиснути *Редагувати символ*. Щоб відкрити вікно властивостей символа (рис. 4.11), треба двічі натиснути на тип піну (напис *Пасивний*). У відкритому вікні треба змінити *Електричний тип на Вихід живлення*.

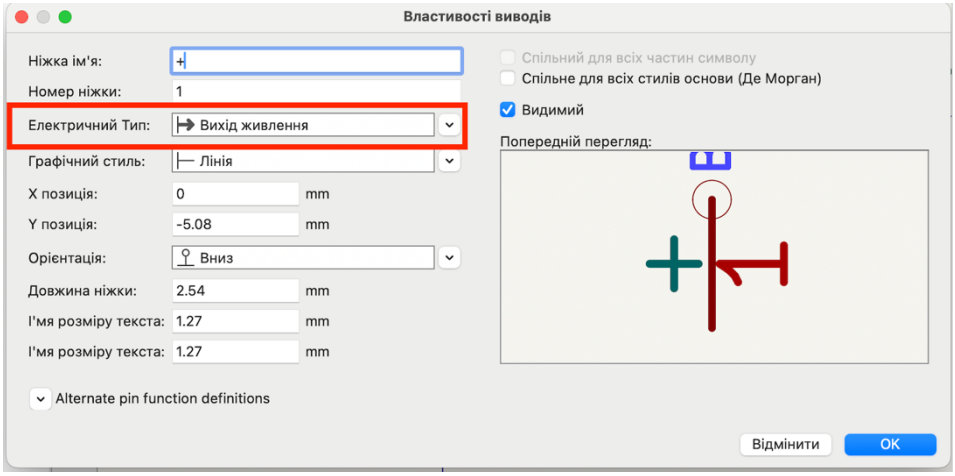


Рисунок 4.11 – Вікно зміни властивостей виводів

VDD/GND можна поставити за допомогою інструменту *Додати символ живлення* на правій панелі або клавішею P, а також додамо символ мікроконтролеру STM32F407VGTx на схему та підключимо живлення відповідно до даташита (схема живлення детально розглянуто у розділі 2).

Розглянемо більш детально електричні типи виводів: *ввід* очікує сигнал ззовні, в*ивід* видає сигнал назовні, *двонаправлений* тип може бути і входом і виходом, *пасивний* вихід не має активної логіки (наприклад, резистор), в*хід живлення* та *вихід живлення* приймають та подають живлення відповідно.

З метою демонстрації базових принципів роботи з цифровими сенсорами, було обрано простий датчик температури та вологості DHT11. Додамо його символ на схему датчик DHT11 [36] (рис. 4.12).

Після цього додамо підпис для з'єднання *DHT11_DATA* за допомогою інструменту *Додати мітку для з'єднання* або клавіші L і розмістимо його на кінці провідника. Для віддзеркалення натиснемо клавішу X.

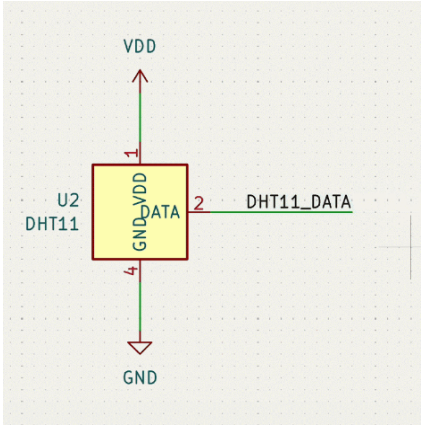


Рисунок 4.12 – Вигляд DHT11 у середовищі KiCad [6]

Розглянемо детальніше даташит DHT11 [36]. У розділі Typical Application (рис. 4.13), бачимо, що датчик потребує підтягуючого резистору номіналом 5 кОм.

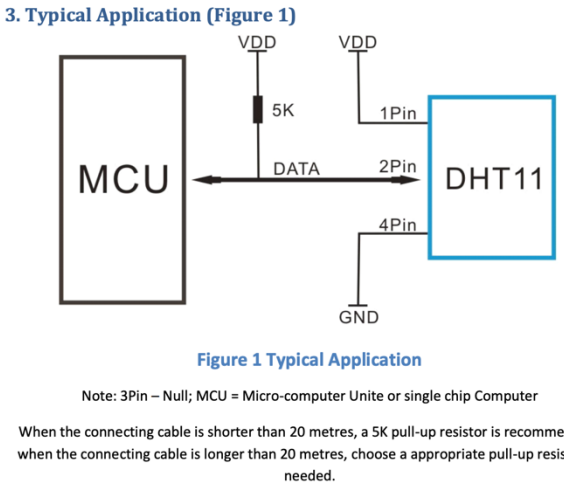


Рисунок 4.13 – Типове підключення DHT11 [36]

Бачимо, що лінія даних потребує підтягуючого резистору номіналом 5 кОм (рис. 4.14).

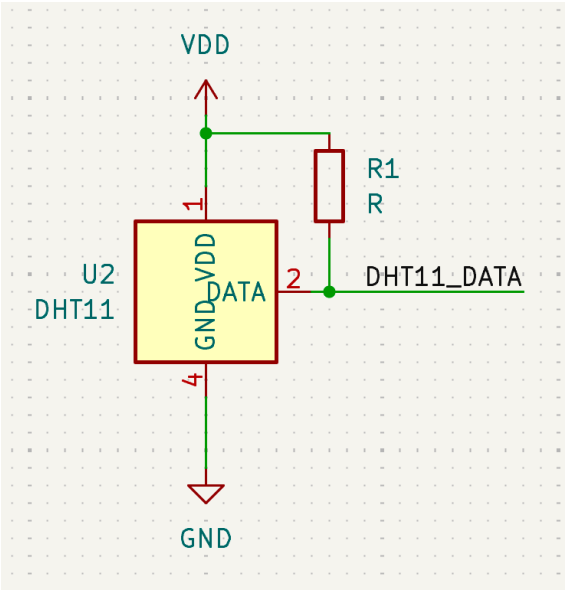


Рисунок 4.14 – Схема підключення DHT11

Тепер додамо ще один напис *DHT11*_DATA, але приєднаємо його до виходу мікроконтролеру, наприклад, PE4 (рис. 4.15).

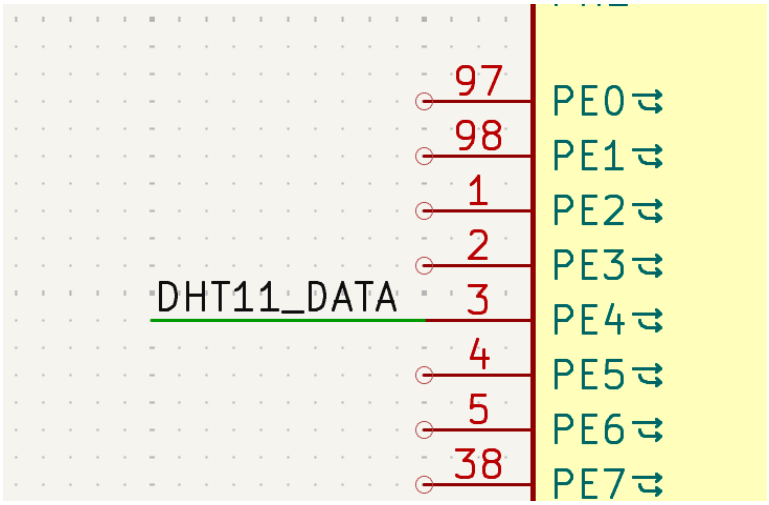


Рисунок 4.15 – Підключення DHT11 до мікроконтролеру

Однак більшість датчиків не мають власних символів, тож розглянемо створення власного символу на прикладі створення типового I2C датчику.

Для цього повернемося до *вікна керування проєктом* та оберемо *Редактор символів*.

Спочатку створимо бібліотеку для проєкту (меню *Файл* > *Створити бібліотеку*), після цього створимо символ. Для цього оберемо новостворену бібліотеку у переліку та оберемо у меню *Файл* > *Новий символ* (рис. 4.16).

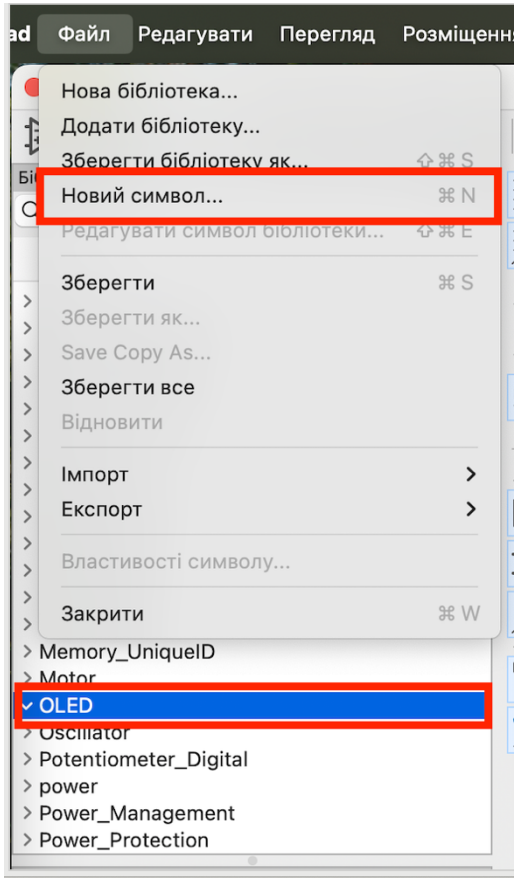


Рисунок 4.16 – Створення нового символу у редакторі символів

Для створення символу ми використаємо лише декілька інструментів: *Додати пін* (клавіша P) та *Додати прямокутник* (рис. 4.17).

Рисунок 4.17 – Панель інструментів редактору символів

Додамо прямоктуник та 4 піни у порядку GND, VCC, SCL, SDA (має відповідати обраному датчику). Обов'язково треба пронумерувати ніжки починаючи із 1. Приклад конфігурації вивода можна побачити на рис. 4.18.

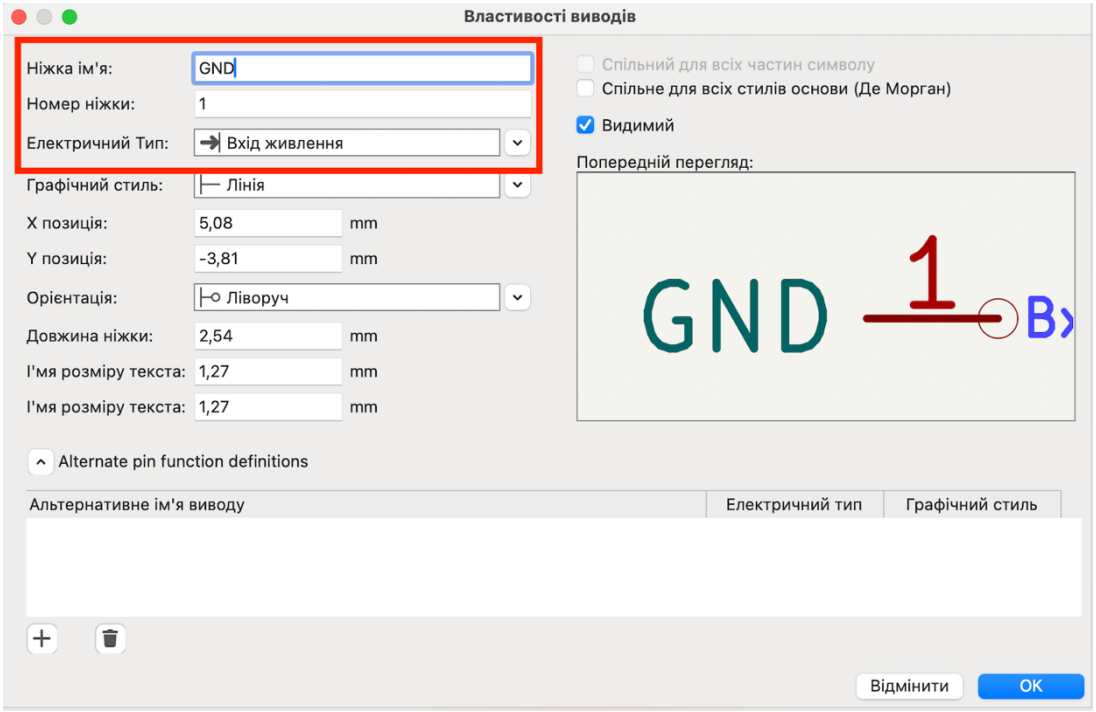


Рисунок 4.18 – Створення виводу

Для GND/VCC правильним електричним типом є *вхід живлення*, для SCL – *ввід*, для SDA – *двонаправлений*, оскільки SCL – *лінія тактування*, мікросхема лише слухає сигнал від контролера, тоді як по SDA мікросхема і приймає і передає дані.

Орієнтацію краще обрати одразу *ліворуч*, але можна буде коректувати у редакторі клавішами R/X. Еталонне зображення символу можна побачити на рис. 4.19.

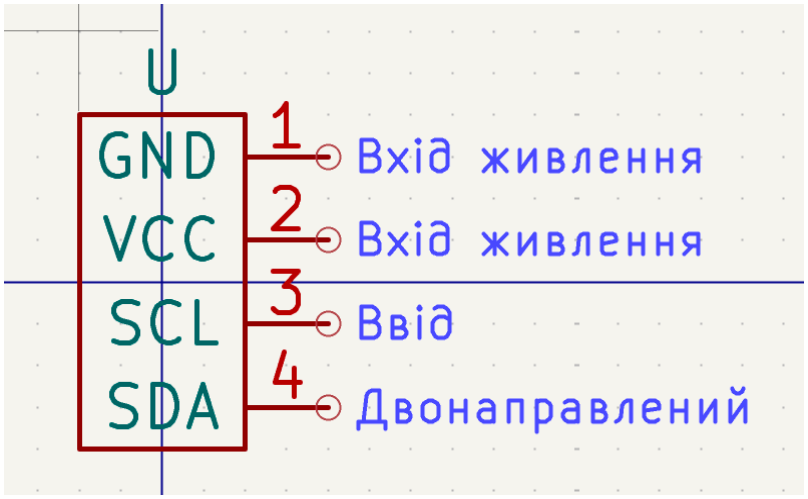


Рисунок 4.19 – Вигляд створеного символу

Щоб додати символ до схеми, потрібно натиснути на *Додати символ в схему* в верхній панелі інструментів (рис. 4.20).

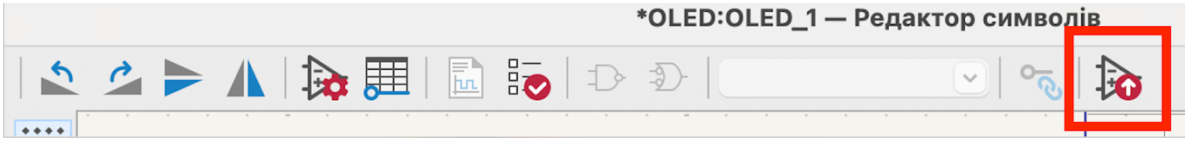


Рисунок 4.20 – Верхня панель інструментів редактору символів

Підключимо живлення, землю, додамо символи для SCL/SDA. Також ці лінії потребують підтягуючих резисторів (рис. 4.21).

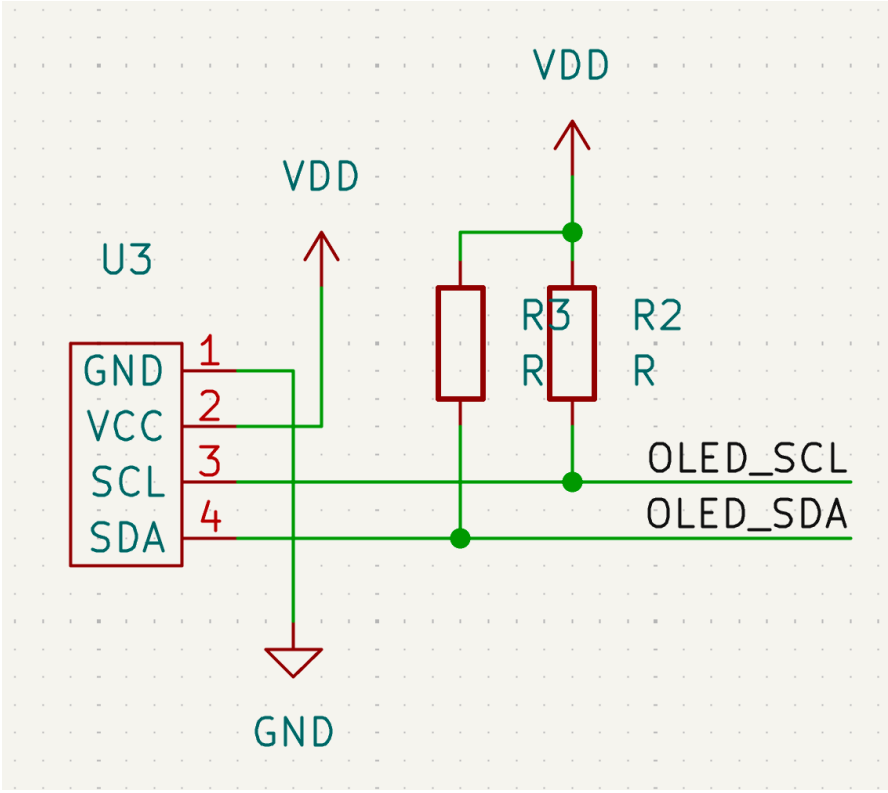


Рисунок 4.21 – Символ типового I2C компоненту

Тепер потрібно розібратися із NRST, BOOT0 та SWD (рис. 4.22).

Символ для SWD потрібно створити із 5 виходами: VDD, GND, SWDIO (двонаправлений), SWCLK (вхід), NRST (вхід).

NRST — це вхід скидання мікроконтролера. Пін має бути підтягнуто резистором номіналом 10 кОм, щоб уникнути випадкового скидання. Натискання кнопки з'єднує пін із землею, що викликає апаратне скидання.

BOOT0 — це вхід, який визначає джерело запуску прошивки при вмиканні живлення. У нормальному режимі підтягнутий до землі (режим запуску з флеш-пам’яті). Натискання кнопки тимчасово подає логічну "1", активуючи системний завантажувач для прошивки мікроконтролера. Схема з цими символами зображена на рис. 4.22.

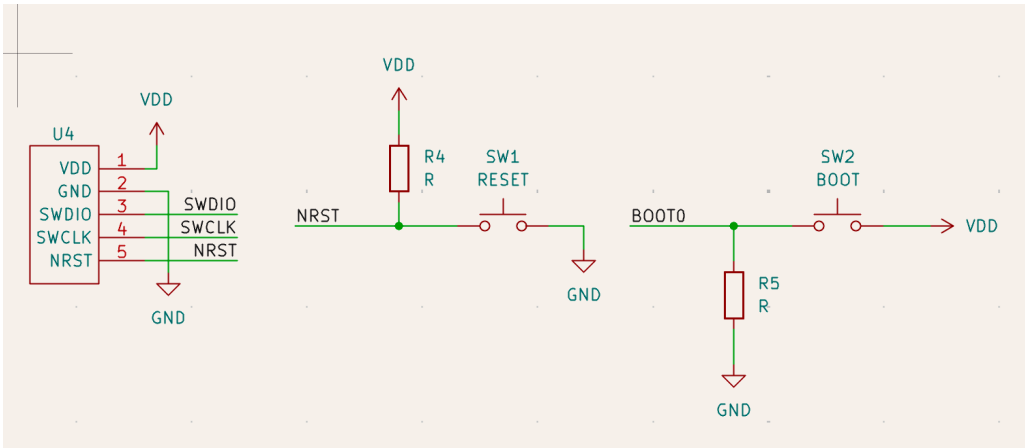


Рисунок 4.22 – Вигляд SWD, NRST, BOOT0

NRST/BOOT0 потрібно підключити до відповідних пінів у мікроконтролері. SWDIO/SWCLK підключаються до PA13/PA14.

Тепер потрібно додати посадкові місця. Для STM32 та DHT11 посадкове місце встановлено у бібліотеці, однак в нас є конденсатори, резистори та власний символ.

Для власних символів ми використаємо просто гребінку (pin headers). Для цього є готовий компонент PinHeader, а саме *PinHeader_1x04_P2.54mm_Vertical* (1 ряд 4 піна, 2.54 мм хід) та *PinHeader_1x05_P2.54mm_Vertical*.

Для батареї оберемо стандартний тримач для Li-Ion 18650 (*BatteryHolder_MPD_BH-18650-PC2*).

Відповідно до даташита нам потрібно по одному керамічному конденсатору на кожен вихід VDD. Однак нам не обов’язково вставляти їх біля самих виходів на принциповій схемі. Ми можемо просто встановити 7 паралельних конденсаторів та підключити їх до VDD/GND (рис. 4.23).

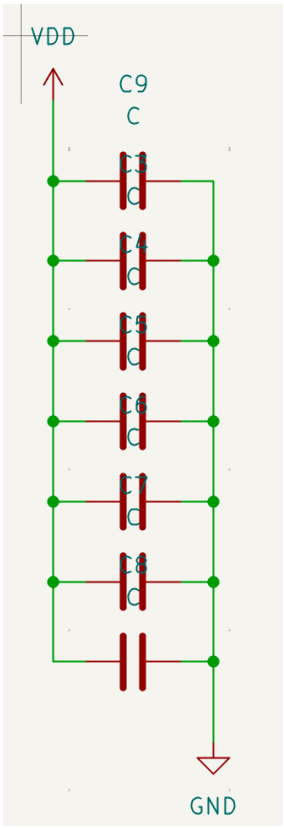


Рисунок 4.23 – Підключення розв’язувальних конденсаторів живлення

Для резисторів та конденсаторів можна встановити однакову модель одразу для усіх. Для цього у меню перейдемо у *Інструменти* > *Редагувати поля*. У відкритому вікні (рис. 4.24) можна поставити посадкові місця одразу для групи резисторів та конденсаторів у полі *Посадкове місце*.

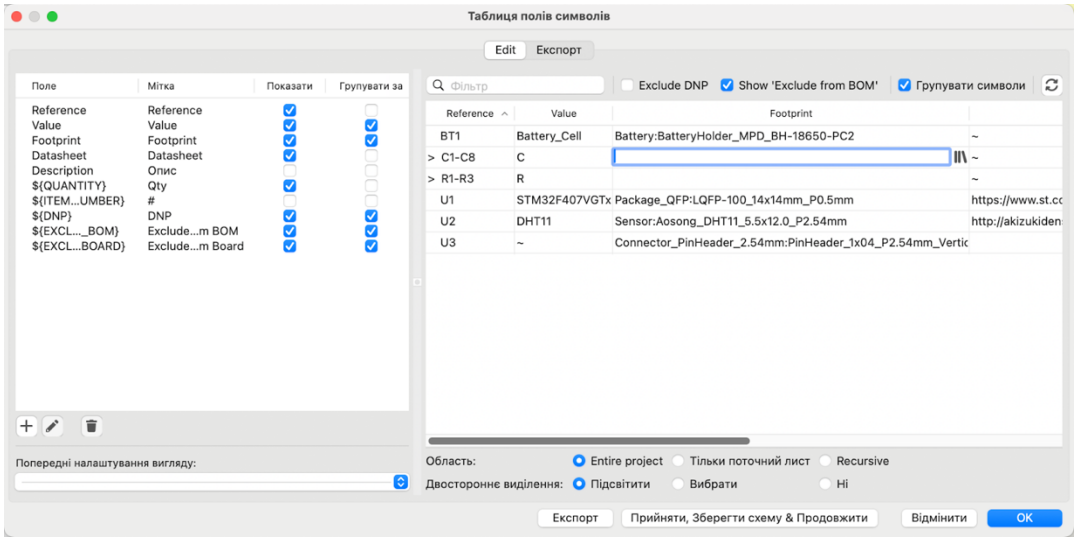


Рисунок 4.24 – Таблиця полів символів схеми

У якості конденсатору оберемо *C_0603_1608Metric_Pad1.08x0.95mm*, у якості резистору *R_0603_1608Metric*. Для кнопок візьмемо *SW_SPST_TL3342*.

Виставимо номінали для резисторів та конденсаторів: для DHT11 – 5k, для VCAP – 2.2 uF, для NRST/BOOT0/I2C пристрою – 10k, для конденсаторів живлення – 100 nF.

Наостанок потрібно розібратися із живленням. Усі елементи схеми живлення розглянуті у розділі 2.

Тепер потрібно створити друковану плату. Повернемося до вікна керування проєктів (див. рис. 4.1) та відкриємо файл із розширенням PCB, щоб відкрити редактор (рис. 4.25).

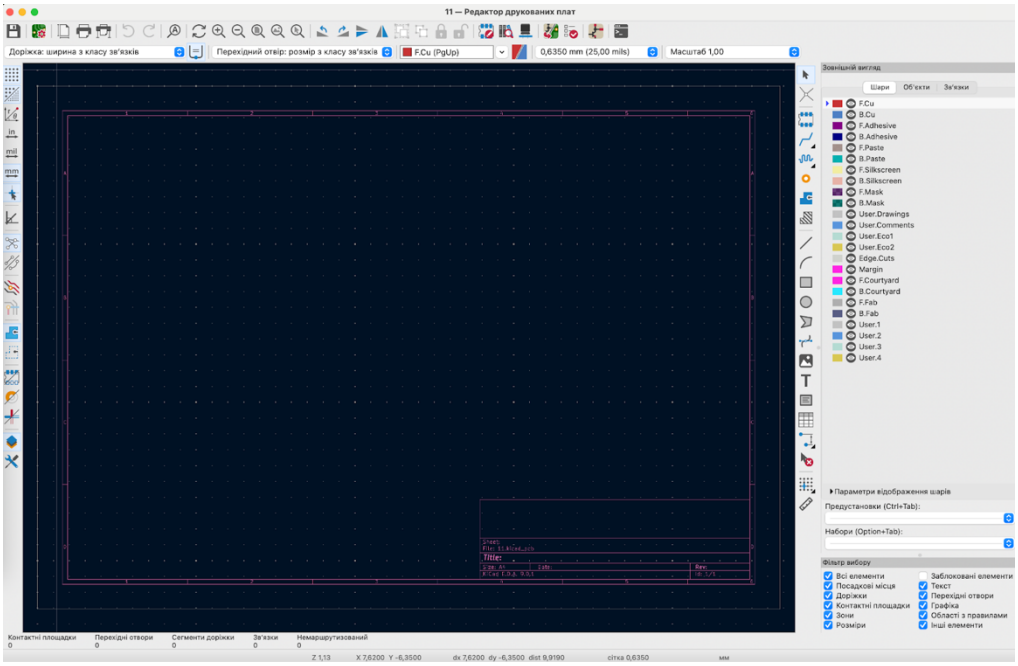


Рисунок 4.25 – Вигляд редактору друкованих плат

Бачимо робочу зону, панелі інструментів та перелік шарів справа.

Перед початком роботи потрібно зробити деякі додаткові налаштування, а також визначити оптимальне компонування.

Одразу змінимо одиниці вимірювання на міли (тисячні долі дюймів, кнопка розташована на лівій панелі інструментів, рис. 4.26).

Рисунок 4.26 – Інструмент для зміни одиниці вимірювання

Перейдемо в меню *Файл* > *Налаштування плати*. Тут нас може зацікавити вкладка *Фізичні параметри* (рис. 4.27), де можна вказати кількість мідних шарів та їх товщину (для нашого прикладу двох шарів цілком достатньо).

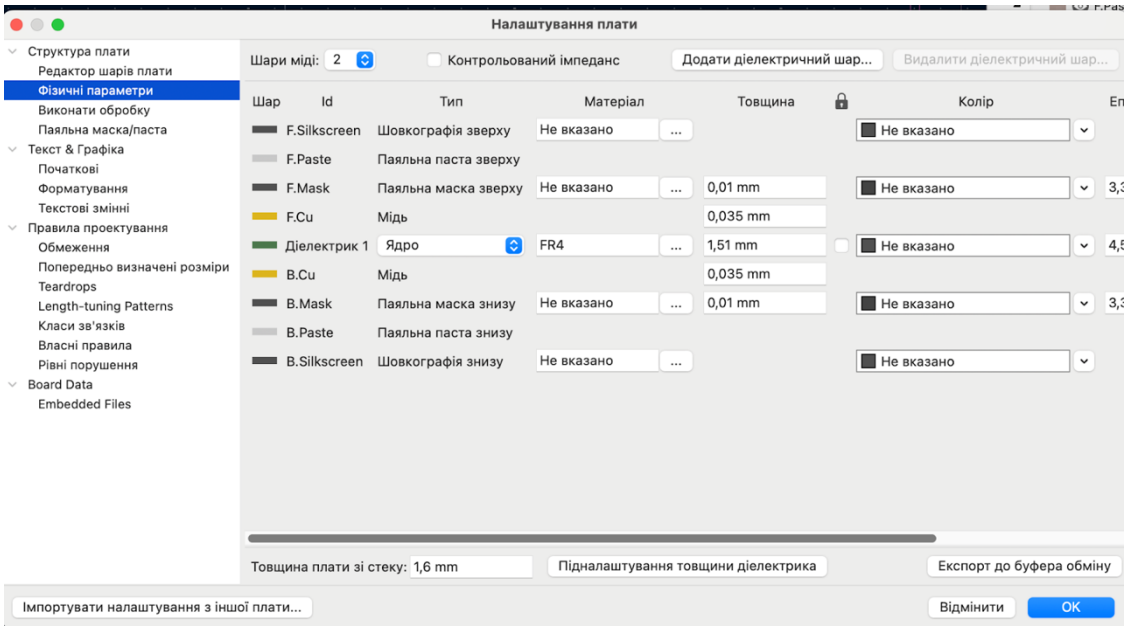


Рисунок 4.27 – Вікно налаштування плати

Перейдемо у вкладку *Правила проєктування* (рис. 4.28). Тут потрібно вказати обмеження виробника. У нашому випадку ми будемо користуватися обмеженнями JLCPCB [37].

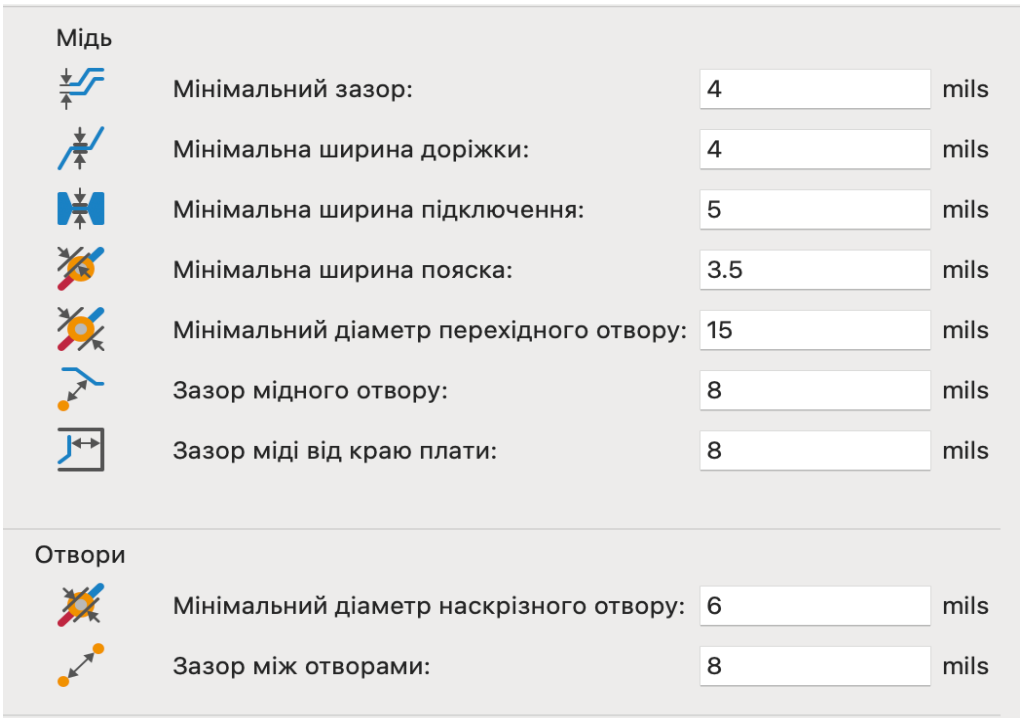


Рисунок 4.28 – Правила проєктування

Перейдемо у розділ *Попередньо визначені розміри*. Тут можна заготувати розміри доріжок та перехідних отворів, щоб швидко змінювати їх через інтерфейс. Додаємо 10, 15, 20 міл для ширини доріжок, для отворів - 24 міл діаметр, 12 міл отвір.

Розглянемо поближче деякі з інструментів. На верхній панелі (рис. 4.29) знаходиться поля для вибору ширини доріжки та розміру отвору (саме з тих розмірів які ми вказали), поле шару та розмір сітки.

Біля поля ширини доріжки знаходиться кнопка, яка робить пріоритет ширини доріжки, яка редагується вищим за обраний у інтерфейсі (наприклад, у нас обрано 20 міл, а доріжка, з якою ми працюємо шириною 10 міл; якщо кнопка активна, то ширина зміниться на 10 міл, що може бути похибкою)

Обираємо ширину доріжки 10 міл, перехідний отвір - 24/12 міл.

Рисунок 4.29 – Верхня панель інструментів

Тепер розглянемо деякі інструменти, які нам знадобляться для створення плати.

У верхній панелі інструментів знаходяться *перевірка правил проєктування* та *оновлення плати зі схеми*. Справа: *додати доріжку*, *додати отвір* та *додати полігон* (рис 4.30).

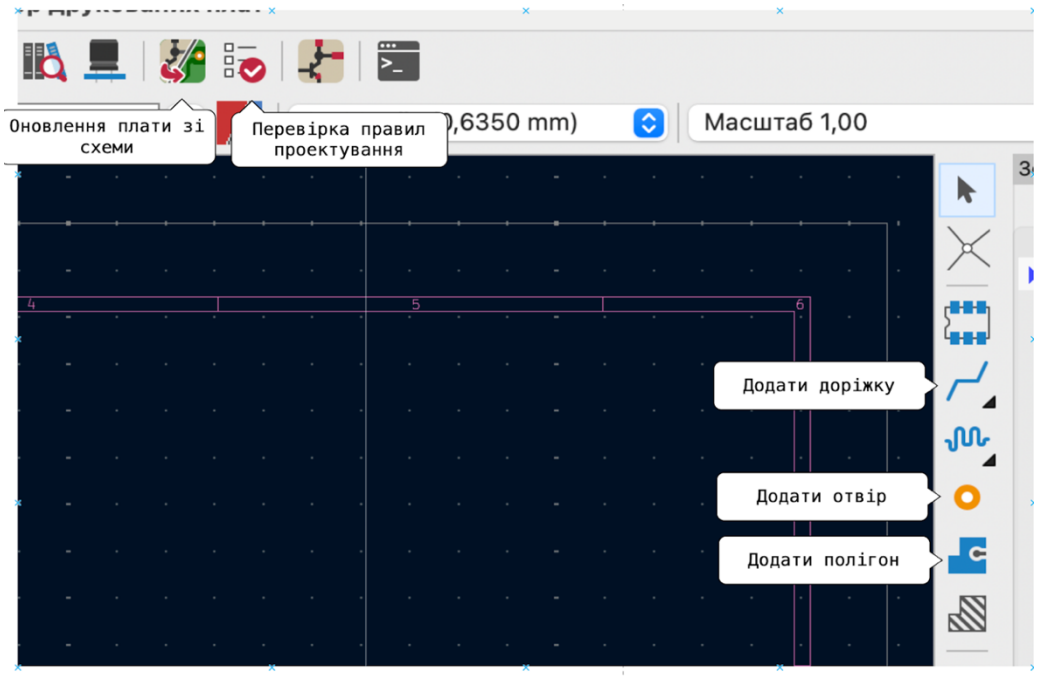


Рисунок 4.30 – Панель інструментів редактору друкованих плат

Спочатку імпортуємо усі компоненти із електричної схеми, натиснувши оновлення плати зі схеми. На цьому етапі також доволі зручно вимкнути текстові написи, відключивши шар F.Fab.

Компоненти можна переміщати мишкою. Усі з'єднання підсвічені синім кольором, Під час з'єднання вони будуть підсвічені. Крім того, на пінах, що мають бути поєднані, будуть однакові назви. Наприклад, на C1 та на піні 73 можна побачити Net-(U1-VCAP_2) - це, очевидно конденсатор VCAP2.

Піни з'єднуються так само, як і у редакторі електричних схем. Обираємо інструмент Додати доріжку та поєднуємо центри пінів мишкою (рис. 4.31).

Рисунок 4.31 – Створення з'єднань

Поєднаємо аналогічним чином усі конденсатори. Від кожного конденсатора, пін живлення якого позначений VDD проведемо невелику лінію, на іншому кінці якої поставимо наскрізний отвір. Через ці отвори буде проходити лінія живлення.

Тепер розберемося із підключенням датчиків. Аналогічно під'єднуємо піни даних, а до цих ліній підводимо резистори. Приклад підключення – на рис. 4.32.

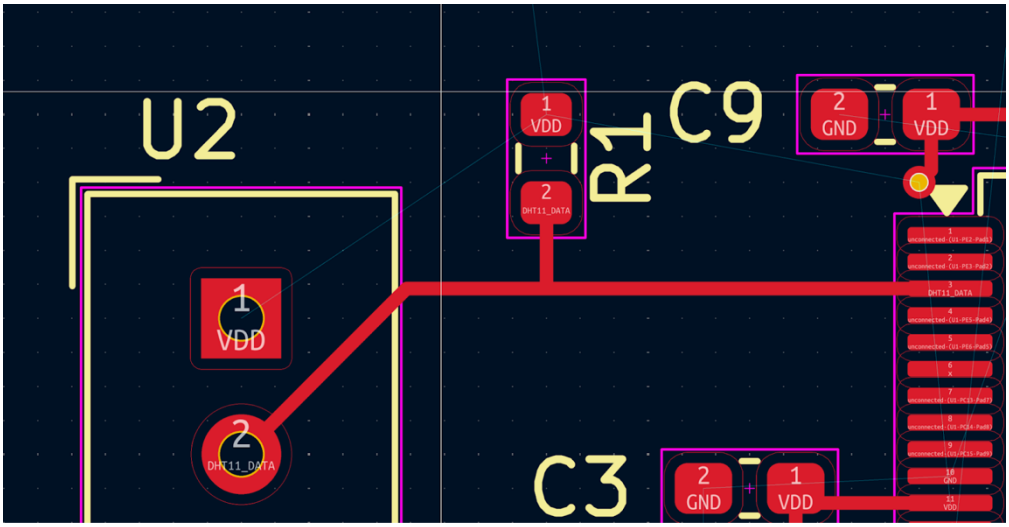


Рисунок 4.32 – Підключення DHT11

Підключимо I2C датчик на іншій стороні плати. Для цього виконаємо наступні дії: змінимо шар на нижній шар міді (B.Cu); це можна зробити, обравши шар у списку зправа або у випадаючому списку шарів зверху, після чого проводимо лінії він виходів датчика поближче до входів на мікроконтролеру (можна проводити прямо під мікроконтролером; щоб зупинити малювання потрібно натиснути Esc), а на кінці лінії ставимо отвір. Після цього переходимо на верхній шар (F.Cu) та з'єднуємо отвір з піном.

Підтягуючи резистори можна приєднати на верхньому слої. Приклад підключення – на рис. 4.33.

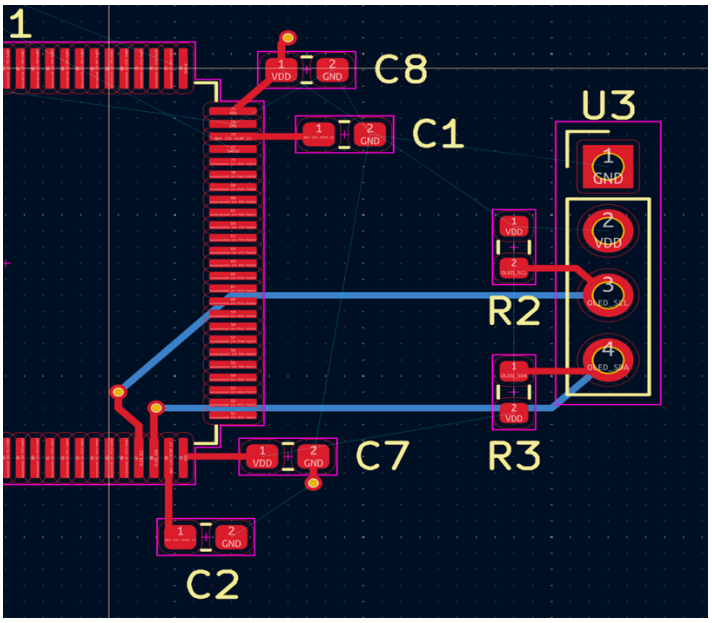


Рисунок 4.33 – Підключення через нижній шар плати

Для того, щоб провести живлення, створимо площину навколо позитивного полюса батарейки на передньому шарі міді. Натиснемо на *Додати полігон*, а потім на плату поруч із полюсом батареї

У відкритому вікні (рис. 4.34) обираємо лінію зв'язку VDD, шар F.Cu, обираємо назву зони, пріоритет обираємо 10. Встановлюємо наступні параметри: зазор – 8 міл, міні мінімальна ширина – 10 міл, тепловий рельєфний відступ – 8 міл, ширина термічного мостика – 15 міл.

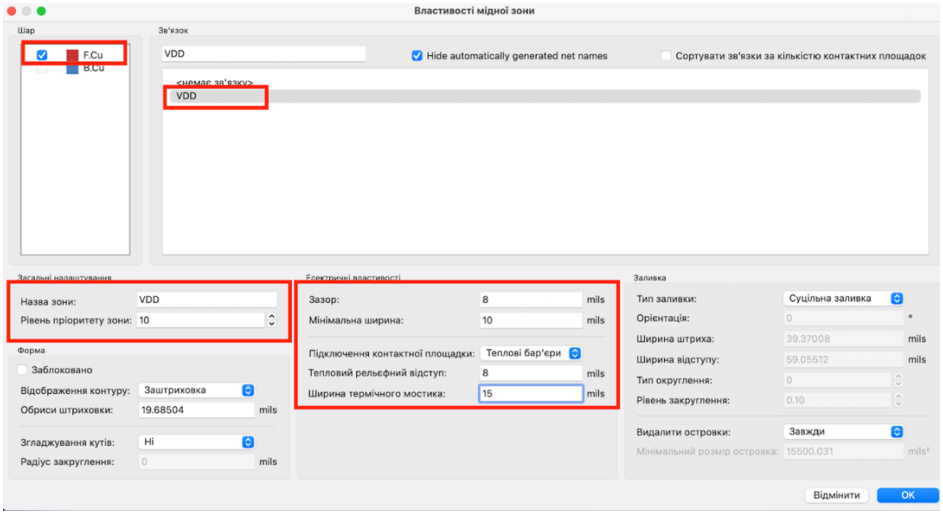


Рисунок 4.34 – Властивості мідної зони

Контур проводиться мишкою. Після того, як контур закритий (це можна зробити або вручну, або через контекстне меню), потрібно натиснути B, щоб залити зону (рис. 4.35).

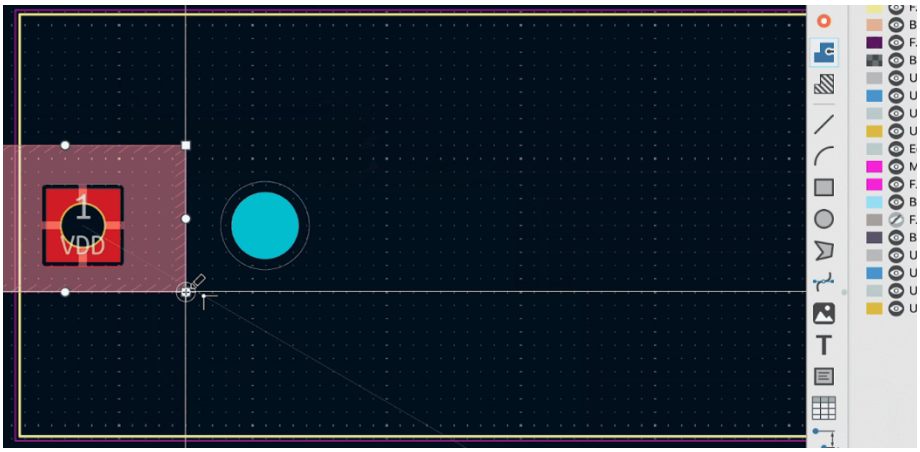


Рисунок 4.35 – Вигляд мідної зони

Після цього, з одного боку VDD додаємо декілька отворів (щоб збільшити площу підключення). Переходимо на нижній шар (B.Cu), обираємо ширину лінію 20 міл та поєднуємо усі отвори, а потім продовжуємо лінію по

усім точкам живлення датчиків та мікроконтролера (в нас є отвори, під'єднані до конденсаторів).

Обов'язково прибираємо пріоритет ширини доріжки (кнопка поруч із вибором ширини у верхній панелі інструментів), щоб коли лінія живлення перетнула якусь лінію на верхньому шарі, вона не змінила товщину!

Тепер додамо контури. Для цього оберемо шар *Edge.Cuts* та використовуючи інструмент *Малювання прямокутника* (на правій панелі інструментів) намалюємо контур навколо плати.

Землю можна додати аналогічно до живлення, однак це можна зробити простіше. Перейдемо на контур F.Cu, виділимо правою клавішею миші контур, Створити з вибраного > Створити зону з виділеного.

У відкритому вікні вводимо такі ж самі технічні параметри, як і для зони VDD, однак лінію обираємо GND та встановлюємо пріоритет 0. Також прибираємо галочку Видалити вихідні об'єкти після перетворення, щоб Edge.Cuts не зникло.

Натискаємо B, щоб залити зону (рис. 4.36).

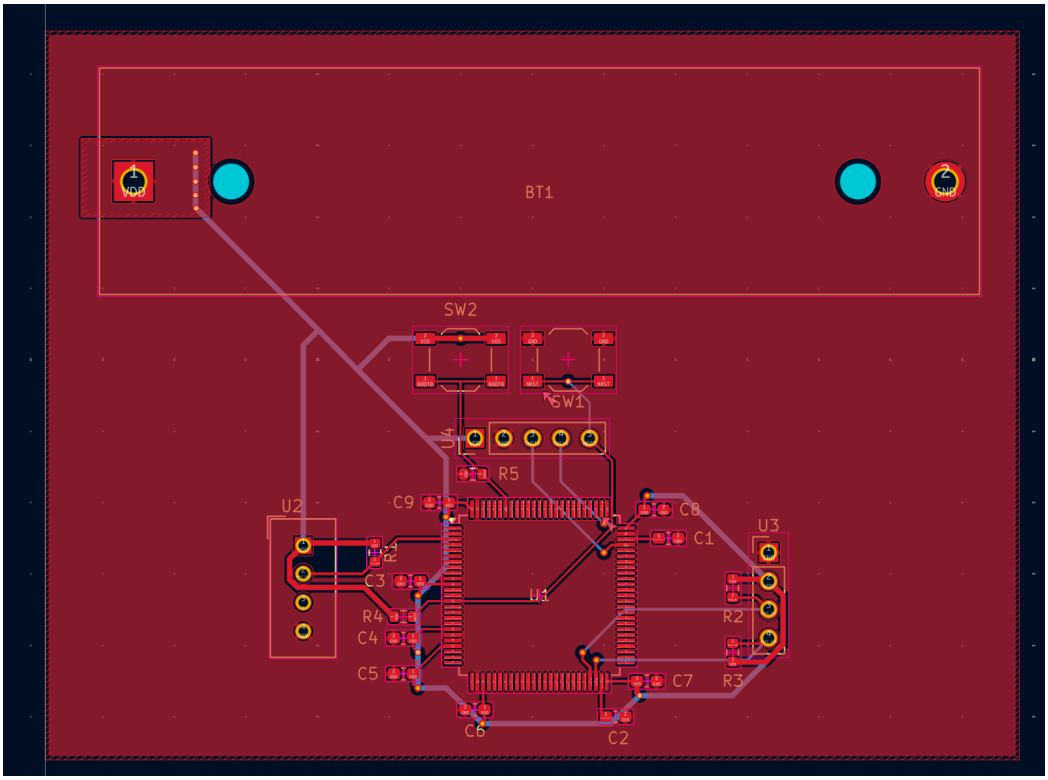


Рисунок 4.36 – Вигляд друкованої плати після додавання зони GND

Запускаємо *перевірку правил проєктування* (див. рис 4.30). На цьому етапі, якщо усі дії виконані вірно, єдиними помилками можуть бути проблеми із зазорами/відстанями/розмірами елементів. У разі успіху, вікно має бути пустим (рис. 4.37).

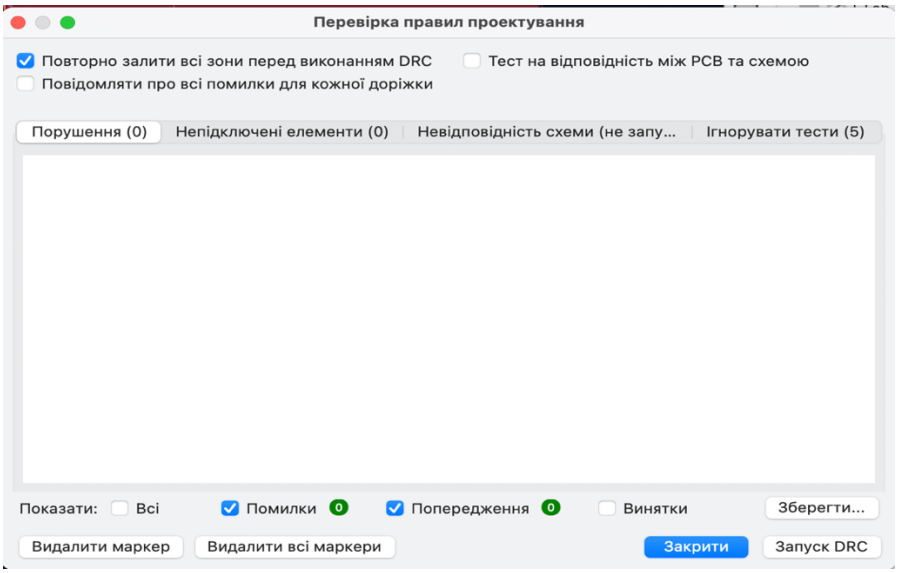


Рисунок 4.37 – Результат перевірки правил проєктування

Тепер плата готова до виробництва. Для створення Gerber та Drill файлів потрібно перейти в розділ меню *Файл* > *Файли виробництва*.

ВИСНОВКИ

У цій роботі було створено навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати, а також спроєктовано автономну метеостанцію.

У першому розділі було проведено аналіз актуальності розробки навчального комплексу, а також розглянуто сучасні середовища для проєктування електронних систем. Було обґрунтовано вибір KiCad 9.0 у якості середовища для навчання студентів.

У другому розділі було розглянуто теорію, на основі якої був розроблений навчальний комплекс: основи електроніки, схемотехніки та програмування STM32. Були надані практичні рекомендації для трасування доріжок, розміщення компонентів, реалізації автономного живлення та пайки елементів

У третьому розділі було спроєктовано автономну метеостанцію, яка включає модуль для вимірювання освітленості (BH1750), модуль ультрафіолетового випромінювання (VEML6070), комбінований датчик для тиску, вологості та тиску (BME280) та датчик для виявлення опадів (FC-37). Дані зчитуються мікросхемою STM32, а потім передаються на сервер через стільниковий модем SIM7600E-H. Було реалізовано енергозберігаючу роботу з використанням STOP-режиму та періодичне пробудження через RTC Wakeup Timer.

У четвертому розділі було протестовано кожен сенсор, роботу стільникового модему та алгоритм дій пристрою. Отримані дані було відправлено на HTTP-сервер, де вони були збережені та проаналізовано. У результаті аналізу даних було підтверджено правильність роботи пристрою.

Отримані результати демонструють практичну доцільність створення навчального комплексу. Виконана робота охоплює усі аспекти проєктування електронних систем: від аналізу вимог до виробництва друкованої плати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a PCB? [Електронний ресурс] – Режим доступу до ресурсу: https://resources.altium.com/p/what-is-a-pcb (дата звернення: 20.05.2025).
2. Altium Designer – професійне середовище для проєктування друкованих плат. [Електронний ресурс] – Режим доступу до ресурсу: https://www.altium.com/altium-designer (дата звернення: 20.05.2025).
3. CircuitMaker – безкоштовне середовище проєктування для студентів та хобістів від Altium. [Електронний ресурс] – Режим доступу до ресурсу: https://circuitmaker.com/ (дата звернення: 20.05.2025).
4. EasyEDA – онлайн-редактор для проєктування схем і друкованих плат. [Електронний ресурс] – Режим доступу до ресурсу: https://easyeda.com/ (дата звернення: 20.05.2025).
5. Eagle PCB Design Software. [Електронний ресурс] – Режим доступу до ресурсу: https://www.autodesk.com/products/eagle/overview (дата звернення: 20.05.2025).
6. KiCad EDA. [Електронний ресурс] – Режим доступу до ресурсу: https://www.kicad.org/ (дата звернення: 20.05.2025).
7. Flux.ai – сучасне середовище для спільного редагування електронних схем у браузері. [Електронний ресурс] – Режим доступу до ресурсу: https://www.flux.ai/ (дата звернення: 20.05.2025).
8. PCB Design: Typical Layout Characteristics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.jpe-innovations.com/precision-point/pcb-design-typical-layout-characteristics/ (дата звернення: 20.05.2025).
9. Via Types. [Електронний ресурс] – Режим доступу до ресурсу: https://www.wevolver.com/article/via-types (дата звернення: 20.05.2025).
10. PCB Manufacturing Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://www.protoexpress.com/kb/pcb-manufacturing-overview/ (дата звернення: 20.05.2025).

11. SMT vs SMD vs THT – Electronics Assembly Techniques. [Електронний ресурс] – Режим доступу до ресурсу: https://www.wevolver.com/article/smt-vs-smd-vs-tht-a-comprehensive-guide-to-electronics-assembly-techniques (дата звернення: 20.05.2025).
12. Як користуватися паяльником — 7 порад. [Електронний ресурс] – Режим доступу до ресурсу: https://www.moyo.ua/ua/news/kak_polzovatsya_payalnikom_7_sovetov.html (дата звернення: 20.05.2025).
13. Electronics Basics: MOSFETs. [Електронний ресурс] – Режим доступу до ресурсу: https://www.electronicsforu.com/technology-trends/learn-electronics/mosfet-basics-working-applications (дата звернення: 20.05.2025).
14. Pull-up Resistor, Pull-down Resistor. [Електронний ресурс] – Режим доступу до ресурсу: https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/ (дата звернення: 20.05.2025).
15. MCP1700 – Low Quiescent Current LDO. [Електронний ресурс] – Режим доступу до ресурсу: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP1700-Low-Quiescent-Current-LDO-20001826E.pdf (дата звернення: 20.05.2025).
16. ETA1096 Datasheet. [Електронний ресурс] – Режим доступу до ресурсу: https://www.eta-semi.com/wp-content/uploads/2022/03/ETA1096_V1.1.pdf (дата звернення: 20.05.2025).
17. DW01A Datasheet. [Електронний ресурс] – Режим доступу до ресурсу: https://hmsemi.com/downfile/dw01a.pdf (дата звернення: 20.05.2025).
18. JLCPCB – Documentation Download. [Електронний ресурс] – Режим доступу до ресурсу: https://jlcpcb.com/api/file/downloadByFileSystemAccessId/8579708037413519360 (дата звернення: 20.05.2025).
19. STM32 Reference Manual (RM0090). [Електронний ресурс] – Режим доступу до ресурсу:

https://www.st.com/resource/en/reference_manual/rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата звернення: 20.05.2025).

20. CMSIS Overview – ARM. [Електронний ресурс] – Режим доступу до ресурсу: https://www.arm.com/technologies/cmsis (дата звернення: 20.05.2025).
21. STM32F4 Discovery Kit User Manual. [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf (дата звернення: 20.05.2025).
22. STM32CubeIDE – інтегроване середовище розробки для мікроконтролерів STM32. [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/en/development-tools/stm32cubeide.html (дата звернення: 20.05.2025).
23. PlatformIO IDE. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.io/ (дата звернення: 20.05.2025).
24. Visual Studio Code. [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/ (дата звернення: 20.05.2025).
25. Keil MDK. [Електронний ресурс] – Режим доступу до ресурсу: https://www.keil.com/download/ (дата звернення: 20.05.2025).
26. IAR Embedded Workbench. [Електронний ресурс] – Режим доступу до ресурсу: https://www.iar.com/ewarm (дата звернення: 20.05.2025).
27. Basics of the I2C Communication Protocol. [Електронний ресурс] – Режим доступу до ресурсу: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ (дата звернення: 20.05.2025).
28. UART – A Hardware Communication Protocol. [Електронний ресурс] – Режим доступу до ресурсу: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html (дата звернення: 20.05.2025).

29. ST-LINK 2.0 — офіційний програматор та інтерфейс налагодження для STM32. [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/en/development-tools/st-link-v2.html (дата звернення: 20.05.2025).
30. TypeScript — мова програмування, що розширює можливості JavaScript за рахунок статичної типізації. [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/ (дата звернення: 20.05.2025).
31. Node.js — середовище виконання JavaScript на стороні сервера. [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/ (дата звернення: 20.05.2025).
32. Express.js — мінімалістичний фреймворк для Node.js. [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/ (дата звернення: 20.05.2025).
33. JavaScript — мова програмування для створення інтерактивних веб-додатків. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 20.05.2025).
34. Matplotlib – бібліотека для побудови графіків у Python. [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/ (дата звернення: 20.05.2025).
35. Python — офіційна документація мови програмування Python. [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/ (дата звернення: 20.05.2025).
36. DHT11 Temperature & Humidity Sensor – Technical Data Sheet. – Електронний ресурс. – Режим доступу: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf – Дата звернення: 27.05.2025.
37. JLCPCB Capabilities — Manufacturing Capabilities Overview. – Електронний ресурс. – Режим доступу: https://jlcpcb.com/capabilities – Дата звернення: 27.05.2025.

ДОДАТОК А

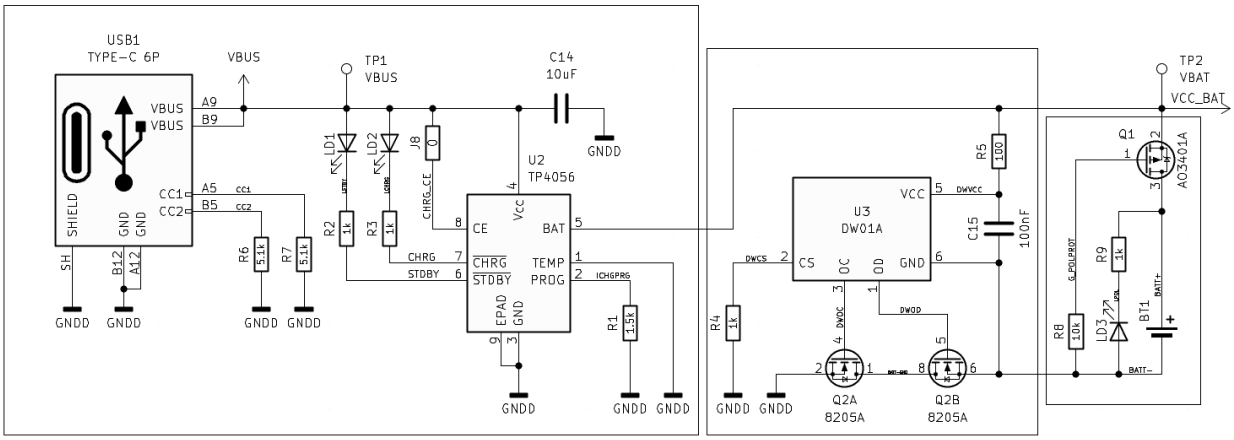
Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

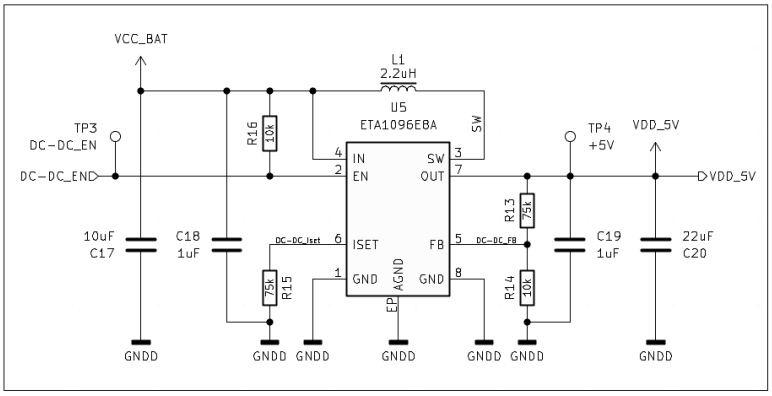
Принципова схема пристрою

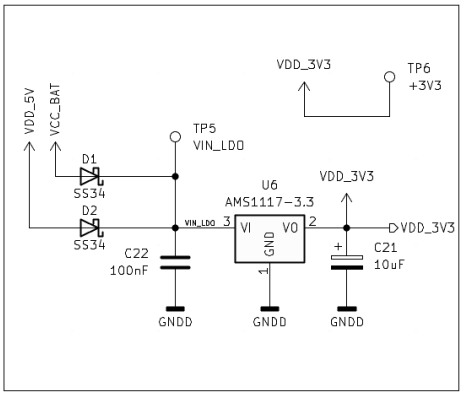
ІАЛЦ.467200.004 ДА

Аркушів 2

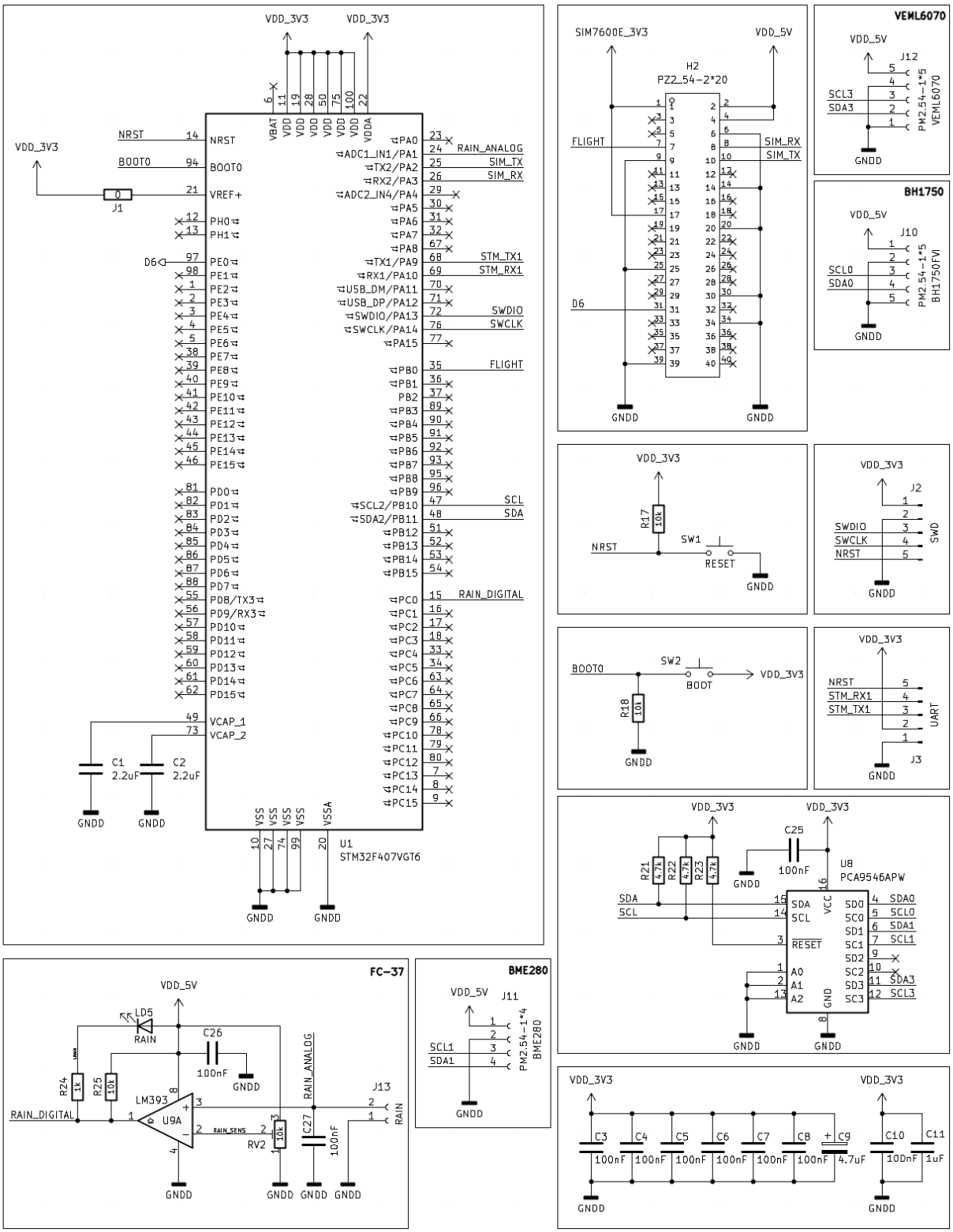
Київ – 2025







					ІАЛЦ.467200.004 ДА	
Зм.	*Арк.*	*№ докум.*	*Підп.*	*Дата*		
Розроб.		*Воробйов А. Р.*			*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати* *Принципова схема пристрою*	*Літ.* / *Арк.* / *Аркушів*: 1 / 2
Перевір.		*Клименко І. А.*				
Реценз.						*НТУУ «КПІ» ФІОТ* *ІО-12*
Н. контр.		*Нікольський С. С.*				
Затверд.						



Зм.	Арк.	№ докум.	Підп.	Дат	ІАЛЦ.467200.004 ДА	Арк. 2

ДОДАТОК Б

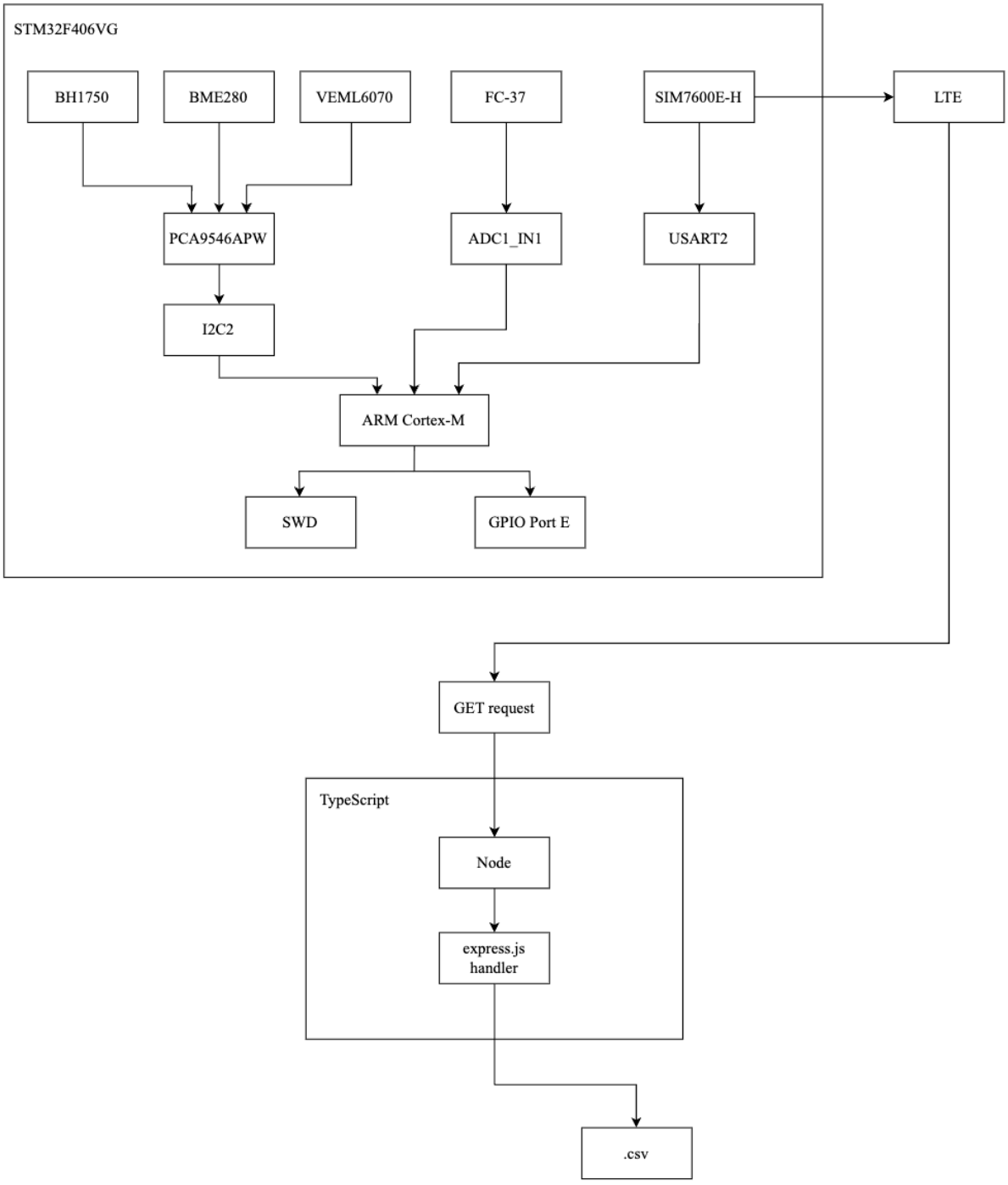
Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

Структурна схема пристрою

ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ – 2025



					ІАЛЦ.467200.005 ДБ		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Воробйов А. Р.			*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати*	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.					
Реценз.							
Н. контр.		Нікольський С. С.			*Структурна схема пристрою*	**НТУУ «КПІ» ФІОТ ІО-12**	
Затверд.							

ДОДАТОК В

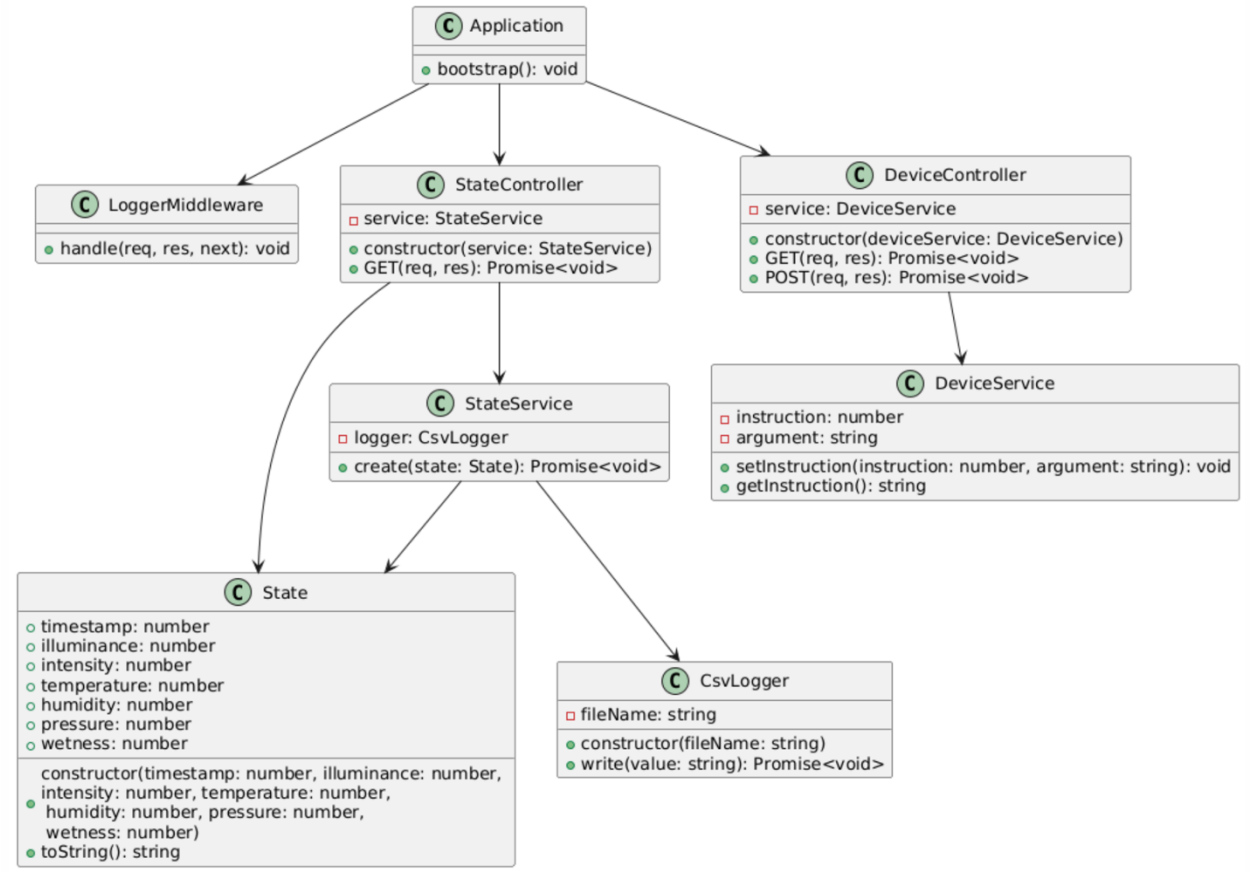
Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

Функціональна схема (діаграма класів)

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ – 2025



					ІАЛЦ.467200.006 ДВ	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розроб.		Воробйов А. Р.			*Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати* *Функціональна схема (діаграма класів)*	Літ. / Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.				
Реценз.						**НТУУ «КПІ» ФІОТ ІО-12**
Н. контр.		Нікольський С. С.				
Затверд.						

ДОДАТОК Г

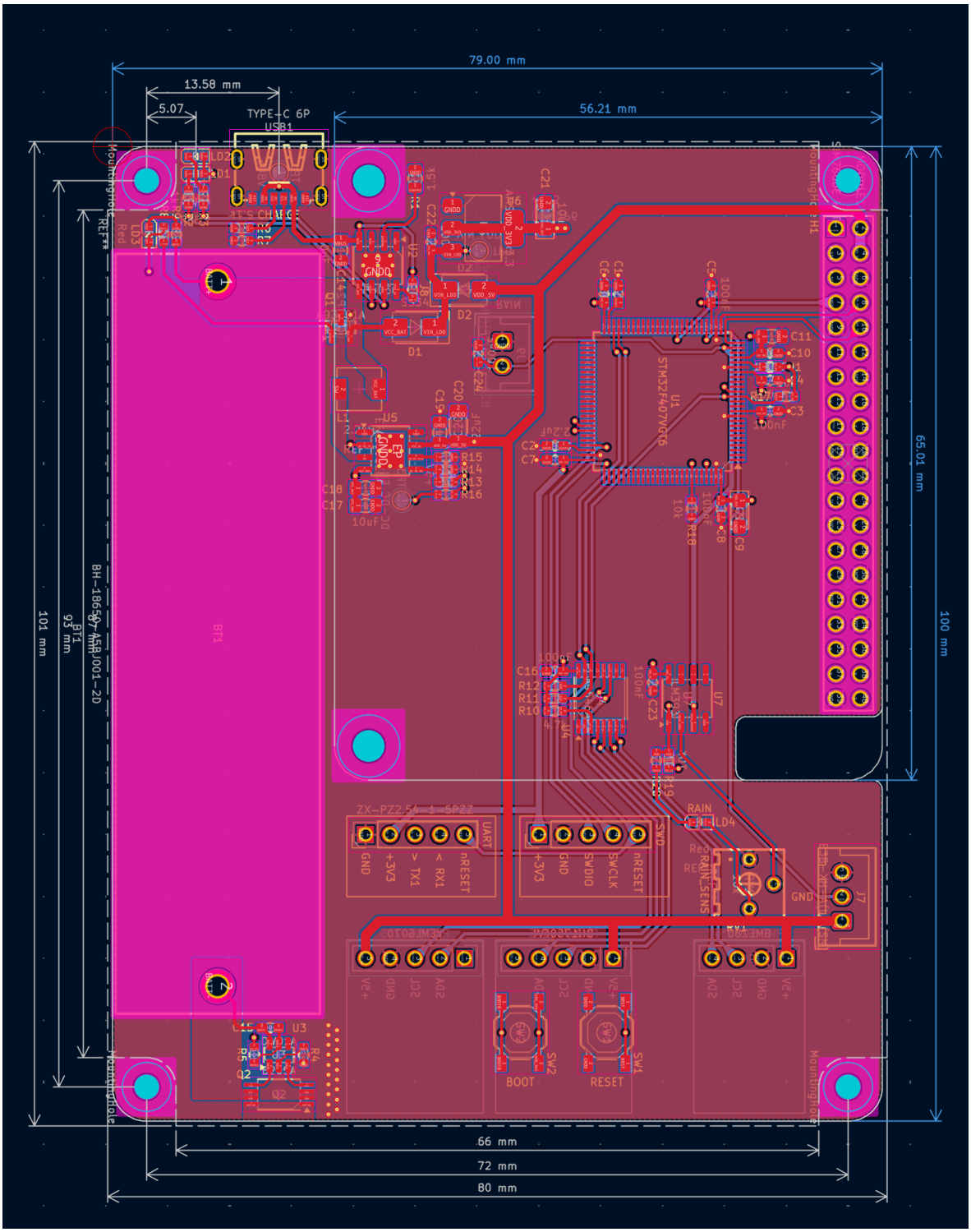
Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

Макет друкованої плати

ІАЛЦ.467200.008 ДГ

Аркушів 1

Київ – 2025



Зм.	Арк.	№ докум.	Підп.	Дата	ІАЛЦ.467200.008 ДГ		
Розроб.		Воробйов А. Р.			Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати Макет друкованої плати	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.					
Реценз.						НТУУ «КПІ» ФІОТ ІО-12	
Н. контр.		Нікольський С. С.					
Затверд.							

ДОДАТОК Ґ

Навчально-методичний комплекс для проєктування електронних систем на базі STM32 із розробкою друкованої плати

Текст програмного коду

ІАЛЦ.467200.007 Д Ґ

Аркушів 16

Київ – 2025

main.c

```
#include <stdbool.h>
#include <inttypes.h>
#include <string.h>
#include <stdarg.h>
// Структура містить калібрувальні коефіцієнти BME280
typedef struct {
    uint16_t t1;
    int16_t t2;
    int16_t t3;
    uint16_t p1;
    int16_t p2;
    int16_t p3;
    int16_t p4;
    int16_t p5;
    int16_t p6;
    int16_t p7;
    int16_t p8;
    int16_t p9;
    uint8_t h1;
    int16_t h2;
    uint8_t h3;
    int16_t h4;
    int16_t h5;
    int8_t h6;
} BME280_Config;
typedef struct {
  char* kind;
  char* argument;
} SIM7600E_Command;
#define RTC_TICKS_PER_SECOND                2048
#define RTC_SECONDS_TO_TICKS(x)     ((x) * RTC_TICKS_PER_SECOND)
#define SECONDS_PER_MINUTE                  60
#define MINUTES_TO_SECONDS(x)       ((x) * SECONDS_PER_MINUTE)
#define IDLE_TIMEOUT                MINUTES_TO_SECONDS(8)
```

```
#define PCA9546A_ADDR                       (0x70 << 1)
#define CH_BH1750                           0
#define CH_BME280                           1
#define CH_VEML6070                         3
#define BH1750_ADDR                         (0x23 << 1)
#define BH1750_H_RESOLUTION_MODE            0x10
#define BH1750_H_RESOLUTION_MODE_DELAY      180
#define BH1750_LUX_FACTOR                   1.2f
#define VEML6070_WRITE_ADDR                 0x38 << 1
#define VEML6070_READ_H_ADDR                0x39 << 1
#define VEML6070_READ_L_ADDR                0x38 << 1
#define VEML6070_1T_NO_ACK_MODE             0x06
#define BME280_ADDR                         (0x76 << 1)
#define BME280_HUMIDITY_REGISTRY            0xF2
#define BME280_MEAS_REGISTRY                0xF4
#define BME280_FILTER_REGISTRY              0xF5
#define BME280_DATA_REGISTRY                0xF7
#define BME280_CONFIG_REGISTRY              0x88
#define BME280_HUMIDITY_OVERSAMPING_X1      0x01
#define BME280_MEAS_OVERSAMPING_X1          0x27
#define BME280_NO_FILTER_500_STANDBY        0x00
#define BME280_HUMIDITY_CONFIG_REGISTRY     0xE1
#define SIM7600E_BUFFER_SIZE                256
#define SIM7600E_NETOPEN                    "AT+NETOPEN"
#define SIM7600E_NETCLOSE                   "AT+NETCLOSE"
#define SIM7500E_SETAPN      "AT+CGDCONT=1,\"IP\",\"%s\""
#define SIM7600E_HTTPINIT                   "AT+HTTPINIT"
#define SIM7600E_SET_URL        "AT+HTTPPARA=\"URL\",\"%s\""
#define SIM7600E_HTTPGET                    "AT+HTTPACTION=0"
#define SIM7600E_HTTPREAD                   "AT+HTTPREAD=0,%d"
#define SIM7600E_STOP                       "AT+CFUN=0"
#define SIM7600E_START                      "AT+CFUN=1"
#define SIM7600E_COMMAND_LENGTH             64
bool sleeping = false;
BME280_Config config;
char* apn = "www.kyivstar.net";
char* endpoint = "http://stm32-debugger.ngrok.io";
// Встановлює канал мультиплексора.
static HAL_StatusTypeDef PCA9546A_SetChannel(uint8_t cid) {
```

```
    uint8_t value = 1 << cid;
    return HAL_I2C_Master_Transmit(&hi2c2, PCA9546A_ADDR, &value, 1,
1000);
}
// Зчитує дані з датчику освітленості
static HAL_StatusTypeDef BH1750_Read(float *illuminance) {
    *illuminance = -2;
    if (PCA9546A_SetChannel(CH_BH1750) != HAL_OK) {
        return HAL_ERROR;
    }
    *illuminance = -1;
    uint8_t command = BH1750_H_RESOLUTION_MODE;
    if (HAL_I2C_Master_Transmit(&hi2c2, BH1750_ADDR, &command, 1,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    HAL_Delay(BH1750_H_RESOLUTION_MODE_DELAY);
    uint8_t data[2];
    if (HAL_I2C_Master_Receive(&hi2c2, BH1750_ADDR, data, 2,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    *illuminance = ((data[0] << 8) | data[1]) / BH1750_LUX_FACTOR;
    return HAL_OK;
}
// Зчитує дані з датчику УФ-випромінювання
static HAL_StatusTypeDef VEML6070_Read(uint32_t *intensity) {
    *intensity = -2;
    if (PCA9546A_SetChannel(CH_VEML6070) != HAL_OK) {
        return HAL_ERROR;
    }
    *intensity = -1;
    uint8_t command = VEML6070_1T_NO_ACK_MODE;
    if (HAL_I2C_Master_Transmit(&hi2c2, VEML6070_WRITE_ADDR, &command,
1, HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    uint8_t left, right;
    if (HAL_I2C_Master_Receive(&hi2c2, VEML6070_READ_L_ADDR, &right,
```

```
1, HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    if (HAL_I2C_Master_Receive(&hi2c2, VEML6070_READ_H_ADDR, &left, 1,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    *intensity = (left << 8) | right;
    return HAL_OK;
}
// Записує дані у регістр BME280
static HAL_StatusTypeDef BME280_Write(uint8_t registry, uint8_t
command) {
    uint8_t data[2] = { registry, command };
    if (HAL_I2C_Master_Transmit(&hi2c2, BME280_ADDR, data, 2,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    return HAL_OK;
}
// Зчитує калібрувальні дані з BME280
static HAL_StatusTypeDef BME280_Init(BME280_Config *config) {
  if (PCA9546A_SetChannel(CH_BME280) != HAL_OK) {
    return HAL_ERROR;
  }
    uint8_t data[26] = {0};
    uint8_t registry = BME280_CONFIG_REGISTRY;
    if (HAL_I2C_Master_Transmit(&hi2c2, BME280_ADDR, &registry, 1,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    if (HAL_I2C_Master_Receive(&hi2c2, BME280_ADDR, data, 26,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    uint8_t hdata[7] = {0};
    registry = BME280_HUMIDITY_CONFIG_REGISTRY;
    if (HAL_I2C_Master_Transmit(&hi2c2, BME280_ADDR, &registry, 1,
HAL_MAX_DELAY) != HAL_OK) {
```

```
        return HAL_ERROR;
    }
    if (HAL_I2C_Master_Receive(&hi2c2, BME280_ADDR, hdata, 7,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    config->t1 = (data[1] << 8) | data[0];
    config->t2 = (data[3] << 8) | data[2];
    config->t3 = (data[5] << 8) | data[4];
    config->p1 = (data[7] << 8) | data[6];
    config->p2 = (data[9] << 8) | data[8];
    config->p3 = (data[11] << 8) | data[10];
    config->p4 = (data[13] << 8) | data[12];
    config->p5 = (data[15] << 8) | data[14];
    config->p6 = (data[17] << 8) | data[16];
    config->p7 = (data[19] << 8) | data[18];
    config->p8 = (data[21] << 8) | data[20];
    config->p9 = (data[23] << 8) | data[22];
    config->h1 = hdata[25];
    config->h2 = (hdata[1] << 8) | hdata[0];
    config->h3 = hdata[2];
    config->h4 = (hdata[3] << 4) | (hdata[4] & 0x0F);
    config->h5 = (hdata[5] << 4) | (hdata[4] >> 4);
    config->h6 = (int8_t)hdata[6];
}
// Реалізує компенсаторну функцію для температури
static void BME280_ReadTemperature(uint8_t *data, float *t, int32_t
*t_fine) {
    int32_t value = ((int32_t)data[3] << 12) | ((int32_t)data[4] << 4)
| (data[5] >> 4);
    int32_t x = ((((value >> 3) - ((int32_t)config.t1 << 1))) *
((int32_t)config.t2)) >> 11;
    int32_t y = (((((value >> 4) - ((int32_t)config.t1)) * ((value >>
4) - ((int32_t)config.t1))) >> 12) * ((int32_t)config.t3)) >> 14;
    *t_fine = x + y;
    *t = ((*t_fine * 5 + 128) >> 8) / 100.0f;
}
// Реалізує компенсаторну функцію для тиску
static void BME280_ReadPressure(uint8_t *data, int32_t t_fine, float
```

```
*p) {
    int64_t x = ((int64_t)t_fine) - 128000;
    int64_t y = x * x * (int64_t)config.p6;
    y = y + ((x * (int64_t)config.p5) << 17);
    y = y + (((int64_t)config.p4) << 35);
    x = ((x * x * (int64_t)config.p3) >> 8) + ((x *
(int64_t)config.p2) << 12);
    x = (((((int64_t)1) << 47) + x)) * ((int64_t)config.p1) >> 33;
    if (x == 0) {
        *p = 0;
        return;
    }
    int64_t z = 1048576 - (((int32_t)data[0] << 12) |
((int32_t)data[1] << 4) | (data[2] >> 4));
    z = (((z << 31) - y) * 3125) / x;
    x = (((int64_t)config.p9) * (z >> 13) * (z >> 13)) >> 25;
    y = (((int64_t)config.p8) * z) >> 19;
    z = ((z + x + y) >> 8) + (((int64_t)config.p7) << 4);
    *p = (float)z / 25600.0f;
}
// Реалізує компенсаторну функцію для вологості
static void BME280_ReadHumidity(uint8_t *data, int32_t t_fine, float
*h) {
    int32_t z = t_fine - 76800;
    z = (((((((int32_t)data[6] << 8) | data[7]) << 14) -
            (((int32_t)config.h4) << 20) - (((int32_t)config.h5) * z))
+
           16384) >>
          15) *
         (((((((z * ((int32_t)config.h6)) >> 10) *
              (((z * ((int32_t)config.h3)) >> 11) + 32768)) >>
             10) +
            2097152) *
               ((int32_t)config.h2) +
           8192) >>
          14));
    z = z - (((((z >> 15) * (z >> 15)) >> 7) * ((int32_t)config.h1))
>> 4);
    z = (z < 0) ? 0 : z;
```

```
    z = (z > 419430400) ? 419430400 : z;
    *h = (z >> 12) / 1024.0f;
}
// Зчитує дані з BME280 (комбінований датчик).
static HAL_StatusTypeDef BME280_Read(float *t, float *h, float *p) {
    *t = -1;
    *h = -1;
    *p = -1;
    if (PCA9546A_SetChannel(CH_BME280) != HAL_OK) {
        return HAL_ERROR;
    }
    if (BME280_Write(BME280_HUMIDITY_REGISTRY,
BME280_HUMIDITY_OVERSAMPING_X1) != HAL_OK) {
        return HAL_ERROR;
    }
    if (BME280_Write(BME280_MEAS_REGISTRY, BME280_MEAS_OVERSAMPING_X1)
!= HAL_OK) {
        return HAL_ERROR;
    }
    if (BME280_Write(BME280_FILTER_REGISTRY,
BME280_NO_FILTER_500_STANDBY) != HAL_OK) {
        return HAL_ERROR;
    }
    HAL_Delay(10);
    uint8_t registry = BME280_DATA_REGISTRY;
    uint8_t data[8] = {0};
    if (HAL_I2C_Master_Transmit(&hi2c2, BME280_ADDR, &registry, 1,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    if (HAL_I2C_Master_Receive(&hi2c2, BME280_ADDR, data, 8,
HAL_MAX_DELAY) != HAL_OK) {
        return HAL_ERROR;
    }
    int32_t t_fine;
    BME280_ReadTemperature(data, t, &t_fine);
    BME280_ReadPressure(data, t_fine, p);
    BME280_ReadHumidity(data, t_fine, h);
    return HAL_OK;
```

```
}
// Форматує строку
static char* sprintf(const char* fmt, ...) {
  va_list args;
  va_start(args, fmt);
  va_list args_safe;
  va_copy(args_safe, args);
  int size = vsnprintf(NULL, 0, fmt, args);
  va_end(args);
  if (size < 0) {
    va_end(args_safe);
    return NULL;
  }
  char* value = (char*)malloc(size + 1);
  if (!value) {
    va_end(args_safe);
    return NULL;
  }
  vsnprintf(value, size + 1, fmt, args_safe);
  va_end(args_safe);
  return value;
}
// Зчитує дані з FC-37 (датчик осадків)
static HAL_StatusTypeDef FC37_Read(uint32_t *surface_wetness) {
    *surface_wetness = -1;
    HAL_ADC_Start(&hadc1);
    if (HAL_ADC_PollForConversion(&hadc1, 10) == HAL_OK) {
        *surface_wetness = HAL_ADC_GetValue(&hadc1);
    } else {
        *surface_wetness = -1;
    }
    HAL_ADC_Stop(&hadc1);
    if (*surface_wetness > 4095 || *surface_wetness < 0) {
        return HAL_ERROR;
    }
    return HAL_OK;
}
// Надсилає AT-команду до модему.
static HAL_StatusTypeDef SIM7600E_Transmit(char* command, char*
buffer, int timeout) {
```

```
  if (buffer != NULL) {
    memset(buffer, 0, SIM7600E_BUFFER_SIZE);
  }
  HAL_StatusTypeDef status = HAL_UART_Transmit(&huart2,
(uint8_t*)sprintf("%s\r\n", command), strlen(command) + 2, buffer ==
NULL ? timeout : 1000);
  if (status != HAL_OK)
  {
    return status;
  }
  if (buffer != NULL)
  {
    status = HAL_UART_Receive(&huart2, (uint8_t*)buffer,
SIM7600E_BUFFER_SIZE - 1, timeout);
    buffer[SIM7600E_BUFFER_SIZE - 1] = '\0';
  }
  return status;
}
// Ініціалізує модем (має бути виконана перед GET-запитом)
static void SIM7600E_Init() {
  char buffer[256];
  SIM7600E_Transmit(sprintf(SIM7500E_SETAPN, apn), buffer, 2500);
  SIM7600E_Transmit(SIM7600E_NETOPEN, buffer, 2500);
  SIM7600E_Transmit(SIM7600E_HTTPINIT, buffer, 2500);
}
// Відправляє дані з сенсорів на сервер.
static void SIM7600E_POST(float i, uint32_t u, float t, float h, float
p, uint32_t w) {
  char buffer[256];
  char *query = sprintf("%s/state/%.2f,%u,%.2f,%.2f,%.2f,%u",
endpoint, i, u, t, h, p, w);
  SIM7600E_Transmit(sprintf(SIM7600E_SET_URL, query), buffer, 3000);
  SIM7600E_Transmit(SIM7600E_HTTPGET, buffer, 3000);
}
// Отримує актуальну команду для модему з серверу
static SIM7600E_Command SIM7600E_FetchCommand() {
  char buffer[256];
  SIM7600E_Command command;
  char *query = sprintf("%s/ctrl", endpoint);
```

```
    SIM7600E_Transmit(sprintf(SIM7600E_SET_URL, query), NULL, 1000);
    SIM7600E_Transmit(SIM7600E_HTTPGET, buffer, 5000);
    SIM7600E_Transmit(sprintf(SIM7600E_HTTPREAD,
SIM7600E_COMMAND_LENGTH), buffer, 1000);
    char *start = strchr(buffer, '#');
    char *end = strrchr(buffer, '#');
    if (start && end && start != end) {
      *end = 0;
      char *value = start + 1;
      command.kind = strtok(value, ",");
      command.argument = strtok(NULL, ",");
    }
    return command;
}
// Отримує з веб-серверу команду та виконує її
static void SIM7600E_FetchAndExecuteCommand() {
  SIM7600E_Command command = SIM7600E_FetchCommand();
  switch (atoi(command.kind)) {
    case 1: NVIC_SystemReset(); break;
    case 2: apn = command.argument; break;
    case 3: endpoint = command.argument; break;
  }
}
// RTC Wake Up callback
void HAL_RTCEx_WakeUpTimerEventCallback(RTC_HandleTypeDef *h){
    sleeping = false;
}
int main(void) {
  // ...
  float illuminance;
  uint32_t intensity;
  float t, h, p;
  uint32_t wetness;
  BME280_Init(&config);
  while (1) {
    // sleeping змінюється у RTC Wake Up callback
    if (!sleeping) {
      sleeping = true;
      BH1750_Read(&illuminance);
```

```
        HAL_Delay(150);
        VEML6070_Read(&intensity);
        HAL_Delay(150);
        FC37_Read(&wetness);
        HAL_Delay(150);
        BME280_Read(&t, &h, &p);
        SIM7600E_Transmit(SIM7600E_START, NULL, 3000);
        HAL_Delay(15000);
        SIM7600E_Init();
        SIM7600E_FetchAndExecuteCommand();
        SIM7600E_POST(illuminance, intensity, t, h, p, wetness);
        SIM7600E_Transmit(SIM7600E_STOP, NULL, 3000);
        HAL_Delay(15000);
      }
      __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
      // Входимо в STOP-режим
      HAL_SuspendTick();
      HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,
PWR_STOPENTRY_WFI);
      HAL_ResumeTick();
      // при виході відновлюємо SystemClock/I2C/UART
      SystemClock_Config();
      HAL_I2C_DeInit(&hi2c2);
      __HAL_RCC_I2C2_CLK_ENABLE();
      MX_I2C2_Init();
      HAL_UART_DeInit(&huart2);
      __HAL_RCC_USART2_CLK_ENABLE();
      MX_USART2_UART_Init();
      HAL_Delay(50);
    }
}
```

index.ts

```
import express from "express"
import fs from "fs/promises"
import statusCodes from "http-status-codes"
import { config } from "dotenv"
import bodyParser from "body-parser"
config()
// Клас для представлення одного зчитування стану з метеостанції
```

```
class State {
  constructor(readonly timestamp: number, readonly illuminance:
number, readonly intensity: number, readonly temperature: number,
readonly humidity: number, readonly pressure: number, readonly
wetness: number) {
  }
  toString() {
    return [this.timestamp, this.illuminance, this.intensity,
this.temperature, this.humidity, this.pressure,
this.wetness].join(',')
  }
}
// Клас для збереження даних у CSV-файл
class CsvLogger {
  constructor(private readonly fileName: string) {}
  async write(value: string) {
    await fs.appendFile(this.fileName, `${value}\n`)
  }
}
// Сервіс для обробки даних з датчиків (запису їх у файл)
class StateService {
  private readonly logger = new CsvLogger("values.csv")
  async create(state: State) {
    await this.logger.write(state.toString())
  }
}
interface ICreateStateReqParams {
  readonly values?: string
}
// Middleware для логування кожного HTTP-запиту
class LoggerMiddleware {
  handle = (req: express.Request, _res: express.Response, next:
express.NextFunction) => {
    console.log(`[${new Date().toISOString()}] ${req.method}
${req.url}`)
    next()
  }
}
// Контролер для обробки запитів із даними з сенсорів
```

```
class StateController {
  constructor(private readonly service: StateService) {}
  GET = async (req: express.Request<ICreateStateReqParams>, res:
express.Response) => {
    if (!req.params.values) {
      res.status(statusCodes.BAD_REQUEST).end()
      return
    }
    const values = req.params.values.toString().split(",").map(Number)
    if (values.length != 6 || values.some(Number.isNaN)) {
      res.status(statusCodes.BAD_REQUEST).end()
      return
    }
    await this.service.create(
      new State(
        new Date().getTime(),
        ...values as [number, number, number, number, number, number],
      )
    )
    res.status(statusCodes.OK).end()
  }
}
// Сервіс для збереження та видачі команд для пристрою
class DeviceService {
  private instruction: number = undefined
  private argument: string = undefined
  setInstruction(instruction: number, argument?: string) {
    this.instruction = instruction
    this.argument = argument
  }
  getInstruction() {
    const instruction = this.instruction
    const argument = this.argument
    if (!instruction) {
      return ""
    }
    this.instruction = undefined
    this.argument = undefined
    return `#${instruction}${!argument ? "" : `,${argument}`}#`
```

```
  }
}
// Контролер для обробки запитів до керування пристроєм
class DeviceController {
  constructor(private readonly service: DeviceService) {
  }
  GET = async (_req: express.Request, res: express.Response) =>{
   res.status(statusCodes.OK).end(this.service.getInstruction())
  }
  POST = async (req: express.Request, res: express.Response) =>{
    this.service.setInstruction(req.body.instruction,
req.body.argument)
    res.status(statusCodes.OK).end()
  }
}
// Основний клас застосунку
class Application {
  bootstrap() {
    const app = express()
    app.use(bodyParser.urlencoded())
    app.use(express.json())
    const stateService = new StateService()
    const stateController = new StateController(stateService)
    const deviceService = new DeviceService()
    const deviceController = new DeviceController(deviceService)
    const middleware = new LoggerMiddleware()
    app.use((req, res, next) => middleware.handle(req, res, next))
    app.get("/state/:values", stateController.GET)
    app.get("/ctrl", deviceController.GET)
    app.post("/ctrl", deviceController.POST)
    app.listen(process.env.PORT)
  }
}
const app = new Application()
app.bootstrap()
```

main.py

```
import pandas as pd
import requests
import matplotlib.pyplot as plt
```

```
import matplotlib.dates as mdates
from datetime import date
import numpy as np
def read_sensor_values():
  values = pd.read_csv("values.csv")
  values["timestamps"] = pd.to_datetime(values["timestamps"],
unit="ms")
  values["hours"] = values["timestamps"].dt.floor("h")
  return values.groupby("hours")[["t", "h", "p",
"w"]].mean().reset_index()

def fetch_api_values():
  start_date = date(2025, 5, 24)
  url = "https://api.open-meteo.com/v1/forecast"
  params = {
    "latitude": 50.9,
    "longitude": 30.45,
    "hourly": "temperature_2m,relative_humidity_2m,pressure_msl",
    "timezone": "auto",
    "past_days": (date.today() - start_date).days,
  }
  past_days = (date.today() - start_date).days
  data = requests.get(url, params=params).json()
  df = pd.DataFrame({
    "timestamps": pd.to_datetime(data["hourly"]["time"]),
    "t": data["hourly"]["temperature_2m"],
    "h": data["hourly"]["relative_humidity_2m"],
    "p": data["hourly"]["pressure_msl"],
  })
  df["hours"] = df["timestamps"].dt.floor("h")
  return df
sensor_values = read_sensor_values()
api_values = fetch_api_values()
start = sensor_values["hours"].min()
end = sensor_values["hours"].max()
api_values = api_values[
  (api_values["timestamps"] >= start)
    & (api_values["timestamps"] <= end)
]
```

```
def rmse(y_true, y_pred):
  return np.sqrt(np.mean((np.array(y_true) - np.array(y_pred)) ** 2))
for col, name in zip(["t", "h", "p"], ["Температура", "Вологість",
"Тиск"]):
  y_true = sensor_values[col].values
  y_pred = api_values[col].values
  error = rmse(y_true, y_pred)
  pct = 100 * error / np.mean(y_pred)
  print(f"{name}: RMSE = {error:.2f} ({pct:.2f} %)")
plt.rcParams["timezone"] = "Europe/Kyiv"
fig, axes = plt.subplots(2, 2, figsize=(14, 8))
def draw_plot(axes, i, col, title):
  r = i // 2
  c = i % 2
  axes[r][c].plot(sensor_values["hours"], sensor_values[col],
label="Сенсор (середнє)")
  if col in api_values:
    axes[r][c].plot(api_values["hours"], api_values[col], label="API",
linestyle="--")
  axes[r][c].set_title(title)
  axes[r][c].legend()
  axes[r][c].grid(True)
draw_plot(axes, 0, "t", "Температура (°C)")
draw_plot(axes, 1, "h", "Вологість (%)")
draw_plot(axes, 2, "p", "Тиск (гПа)")
draw_plot(axes, 3, "w", "Тиск (гПа)")
for row in axes:
  for axis in row:
    axis.xaxis.set_major_formatter(mdates.DateFormatter('%d.%m
%H:%M'))
    axis.tick_params(axis='x', rotation=45)
plt.tight_layout()
plt.show()
```