МІНІСТЕРСТВО ОСВІТИ І НАУКИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА
Навчально-науковий Інститут енергетичної,
інформаційної та транспортної інфраструктури
Кафедра автоматизації та комп'ютерно-інтегрованих технологій

РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

на тему Автоматизація тестування
програмного забезпечення мікроконтролерних систем

Виконав: здобувач вищої освіти
3 курсу, групи Сінж 2023-1у
напряму підготовки (спеціальності)
174 – Автоматизація, комп'ютерно-інтегровані
технології та робототехніка
Малохвей М.О.
(прізвище та ініціали)

Керівник доц. каф. АКІТ, к.т.н., Аксьонов Є. О.
(прізвище та ініціали, наук. ступ., вчене звання)

Рецензент к.т.н., Ківіренко О. Б.
(прізвище та ініціали, наук. ступ., вчене звання)

Харків – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

Навчально-науковий Інститут енергетичної,

інформаційної та транспортної інфраструктури

Кафедра автоматизації та комп'ютерно-інтегрованих технологій

Освітньо-кваліфікаційний рівень – бакалавр

Галузі знань 17 «Електроніка, автоматизація та електронні комунікації»

Спеціальність 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»

ЗАТВЕРДЖУЮ
Завідувач кафедри АКІТ

_____ **БАРАНОВ О. О.**

«19» червня 2026 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Малохвею Микиті Олександровичу

(прізвище, ім'я, по батькові здобувача вищої освіти)

1. Тема роботи Автоматизація тестування програмного забезпечення мікроконтролерних систем

Керівник роботи Аксьонов Євген Олександрович, к.т.н., доц. каф. АКІТ

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом університету від « 22 » травня 2026 року № 440-03

2. Строк подання роботи здобувачем вищої освіти « 15 » червня 2026 р.

3. Вихідні дані до роботи Існуючі засоби автоматизованого тестування, література із створення систем автоматизованого тестування.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Вступ, аналіз стану проблеми, проєктування системи автоматизованого тестування, розробка прототипу системи і його дослідження, охорона праці, висновки, перелік використаних джерел, додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентація.

6. Консультанти розділів проєкту (роботи)

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Аналіз стану проблеми	Аксьонов Є. О., к.т.н., доцент каф. АКІТ	22.05.2026	22.05.2026
Проєктування системи автоматизованого тестування	Аксьонов Є. О., к.т.н., доцент каф. АКІТ	29.05.2026	29.05.2026
Розробка прототипу системи і його дослідження	Аксьонов Є. О., к.т.н., доцент каф. АКІТ	29.05.2026	29.05.2026
Охорона праці	Малишева В. В., доцент каф. ОПтаБЖД	26.05.2026	26.05.2026

7. Дата видачі завдання « 22 » травня 2026 р.

Керівник Аксьонов Є. О.
(підпис)

Завдання прийняв до виконання Малохвей М.О.
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Розробка 1 розділу «Аналіз стану проблеми»	01.06.2026	
2	Розробка 2 розділу «Проєктування системи автоматизованого тестування»	08.06.2026	
3	Розробка 3 розділу «Розробка прототипу системи і його дослідження»	15.06.2026	
4	Розробка 4 розділу «Охорона праці»	15.06.2026	
5	Нормоконтроль	17.06.2026	
6	Рецензування	20.06.2026	
7	Захист на ДЕК	25.06.2026	

Здобувач вищої освіти Малохвей М.О.
(підпис)

Керівник Аксьонов Є. О.
(підпис)

РЕФЕРАТ

Автоматизація тестування програмного забезпечення мікроконтролерних систем – Малохвей Микита Олександрович, дипломна робота бакалавра, Харків, Харківський національний університет міського господарства імені О. М. Бекетова, кількість сторінок 81, кількість таблиць 12, кількість рисунків 17, кількість джерел літератури 41.

Дипломна робота присвячена розробці автоматизованої системи тестування програмного забезпечення мікроконтролерів, що забезпечує підвищення ефективності налагодження мікроконтролерних систем.

ЦІЛЬ РОБОТИ: підвищення ефективності тестування і налагодження мікрококонтролерних систем шляхом створення спеціалізованого програмного засобу автоматизованого тестування

ОБ'ЄКТ ДОСЛІДЖЕННЯ: процес тестування програмного забезпечення мікроконтролерних систем.

ПРЕДМЕТ ДОСЛІДЖЕННЯ: методи та засоби автоматизованого аналізу програмного коду та структури мікроконтролерних проєктів, створених у середовищі Wokwi

ЗАДАЧІ ДОСЛІДЖЕННЯ: провести аналіз сучасних мікроконтролерів та сфер їх застосування; дослідити існуючі методи розробки, налагодження та тестування програмного забезпечення embedded-систем; виконати аналіз сучасних засобів автоматизованого тестування та моделювання мікроконтролерних проєктів; сформулювати вимоги до системи автоматизованого тестування; розробити структуру та архітектуру програмного забезпечення; реалізувати прототип системи автоматизованого аналізу проєктів Wokwi.

МЕТОДИ ДОСЛІДЖЕННЯ: методи аналізу наукової та технічної літератури, методи структурного аналізу програмного забезпечення, методи статичного аналізу програмного коду, методи обробки структурованих даних та методи експериментального дослідження програмних систем.

КЛЮЧОВІ СЛОВА**:** автоматизація, тестування, програма, код, програмне забезпечення, мікроконтролерні системи.

ABSTRACT

Automation of Software Testing for Microcontroller-Based Systems – Malokhvey Mykyta Oleksandrovych Bachelor's Thesis, Kharkiv, O. M. Beketov National University of Urban Economy in Kharkiv, number of pages 81, number of tables 12, number of figures 17, number of references 41.

The thesis is devoted to the development of an automated system for testing the software of microcontrollers, which ensures an increase in the efficiency of debugging microcontroller systems.

AIM OF THE THESIS: increasing the efficiency of testing and debugging microcontroller systems by creating a specialized software tool for automated testing

OBJECT OF STUDY: the process of testing the software of microcontroller systems.

SUBJECT OF STUDY: methods and means of automated analysis of software code and the structure of microcontroller projects created in the Wokwi environment

OBJECTIVES: analyze modern microcontrollers and their areas of application; to investigate the existing methods of development, debugging and testing of embedded system software; perform an analysis of modern means of automated testing and modeling of microcontroller projects; formulate requirements for the automated testing system; develop the structure and architecture of the software; implement a prototype of the Wokwi automated project analysis system.

RESEARCH METHODS: methods of analysis of scientific and technical literature, methods of structural analysis of software, methods of static analysis of program code, methods of processing structured data and methods of experimental research of software systems.

KEY WORDS**:** automation, testing, program, code, software, microcontroller systems.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АЦП (ADC) - Analog-to-Digital Converter — аналого-цифровий перетворювач;

API - Application Programming Interface — інтерфейс програмування застосунків;

GPIO - General Purpose Input/Output — універсальні входи та виходи мікроконтролера;

GUI - Graphical User Interface — графічний інтерфейс користувача;

HTML - HyperText Markup Language — мова розмітки гіпертекстових документів;

IDE - Integrated Development Environment — інтегроване середовище розробки;

IoT - Internet of Things — Інтернет речей;

JSON - JavaScript Object Notation — формат структурованих даних;

MCU - Microcontroller Unit — мікроконтролер;

ПЗ – Програмне забезпечення;

МК – Мікроконтролер;

ШІМ (PWM) - Pulse Width Modulation — широтно-імпульсна модуляція;

UART - Universal Asynchronous Receiver-Transmitter — послідовний інтерфейс обміну даними;

SPI - Serial Peripheral Interface — послідовний периферійний інтерфейс;

I2C - Inter-Integrated Circuit — послідовний інтерфейс зв’язку;

Wokwi - Онлайн-середовище моделювання мікроконтролерних систем;

Arduino IDE - Середовище розробки для платформи Arduino;

PyQt6 - Бібліотека для створення графічних інтерфейсів мовою Python;

ZIP - Формат архівування файлів;

Firmware - Програмне забезпечення мікроконтролерної системи.

ЗМІСТ

ВСТУП

Актуальність теми роботи. Сучасний розвиток мікроконтролерних систем та embedded-технологій супроводжується постійним зростанням кількості пристроїв, що використовуються в системах автоматизації, промисловості, робототехніці, Інтернеті речей та побутовій електроніці. Збільшення складності програмного забезпечення мікроконтролерних систем підвищує вимоги до його якості, надійності та коректності функціонування. Одним із важливих етапів розробки є тестування програмного забезпечення, яке дозволяє виявляти помилки та забезпечувати відповідність роботи системи встановленим вимогам.

У більшості випадків перевірка мікроконтролерних проєктів виконується вручну шляхом аналізу програмного коду та електронної схеми, що потребує значних витрат часу та залежить від людського фактора. Використання сучасних середовищ моделювання відкриває можливості для автоматизації процесів контролю та оцінювання результатів розробки. У зв'язку з цим розробка системи автоматизованого тестування програмного забезпечення мікроконтролерних систем є актуальним завданням, спрямованим на підвищення ефективності перевірки проєктів та покращення якості програмного забезпечення.

Метою даної роботи є підвищення ефективності тестування і налагодження мікрококонтролерних систем шляхом створення спеціалізованого програмного засобу автоматизованого тестування

Завдання дослідження.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз сучасних мікроконтролерів та сфер їх застосування;
- дослідити існуючі методи розробки, налагодження та тестування програмного забезпечення embedded-систем
- виконати аналіз сучасних засобів автоматизованого тестування та моделювання мікроконтролерних проєктів
- сформулювати вимоги до системи автоматизованого тестування
- розробити структуру та архітектуру програмного забезпечення
- реалізувати прототип системи автоматизованого аналізу проєктів Wokwi.

Об'єктом дослідження є процес тестування програмного забезпечення мікроконтролерних систем.

Предметом дослідження є методи та засоби автоматизованого аналізу програмного коду та структури мікроконтролерних проєктів, створених у середовищі Wokwi.

Під час виконання роботи використовувалися методи аналізу наукової та технічної літератури, методи структурного аналізу програмного забезпечення, методи статичного аналізу програмного коду, методи обробки структурованих даних та методи експериментального дослідження програмних систем.

Практичне значення роботи полягає у розробці програмного засобу, який дозволяє автоматизувати процес перевірки мікроконтролерних проєктів, створених у середовищі Wokwi. Розроблена система може використовуватися у навчальному процесі для контролю виконання лабораторних та практичних робіт, а також для первинного аналізу програмного коду та структури електронних схем.

Дипломна робота складається зі вступу, чотирьох розділів, загальних висновків, переліку використаних джерел та додатків. У першому розділі проведено аналіз предметної області та існуючих підходів до тестування мікроконтролерних систем. Другий розділ присвячений проєктуванню системи автоматизованого тестування. У третьому розділі описано розробку програмного прототипу та результати його дослідження. У четвертому розділі розглянуто питання охорони праці та безпеки під час розробки і тестування програмного забезпечення мікроконтролерних систем.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ

1.1 Мікроконтролерні системи

1.1.1 Мікроконтролер як обчислювальний пристрій

Мікроконтролери (МК) є основою значної частини сучасних embedded-систем та широко використовуються у різноманітних електронних пристроях. Вони застосовуються у побутовій техніці, системах автоматизації, автомобільній електроніці, робототехніці, медичному обладнанні, системах побутового моніторингу та інших сферах. Завдяки компактності, низькому енергоспоживанню та можливості виконання спеціалізованих задач у режимі реального часу МК стали одним із ключових елементів сучасних цифрових систем [1]. На рис. 1.1 представлений типовий МК «Atmega», який є одним з домінуючих у сучасному ринку, та зустрічаються у великій кількості пристроїв різного призначення.

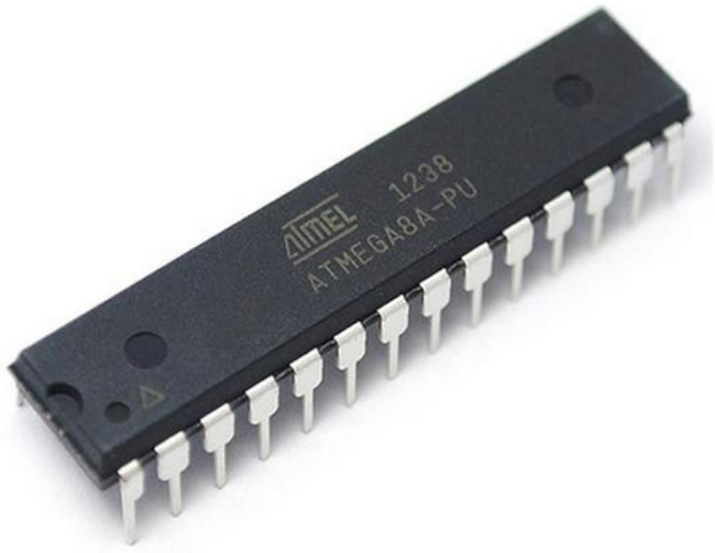

Рисунок 1.1 – Мікроконтролер «Atmega»

МК являє собою спеціалізований обчислювальний пристрій, який містить у межах одного кристалу (рис. 1.2) центральний процесор, оперативну пам’ять, постійну пам’ять програм та периферійні модулі. На відміну від універсальних комп’ютерних систем, МК орієнтовані на виконання конкретних функцій

керування та взаємодії з фізичними пристроями. До складу сучасних МК входять модулі введення-виведення GPIO, таймери, аналого-цифрові та цифро-аналогові перетворювачі, інтерфейси UART, SPI, I2C, CAN, Wi-Fi, Bluetooth тощо [2, 20].

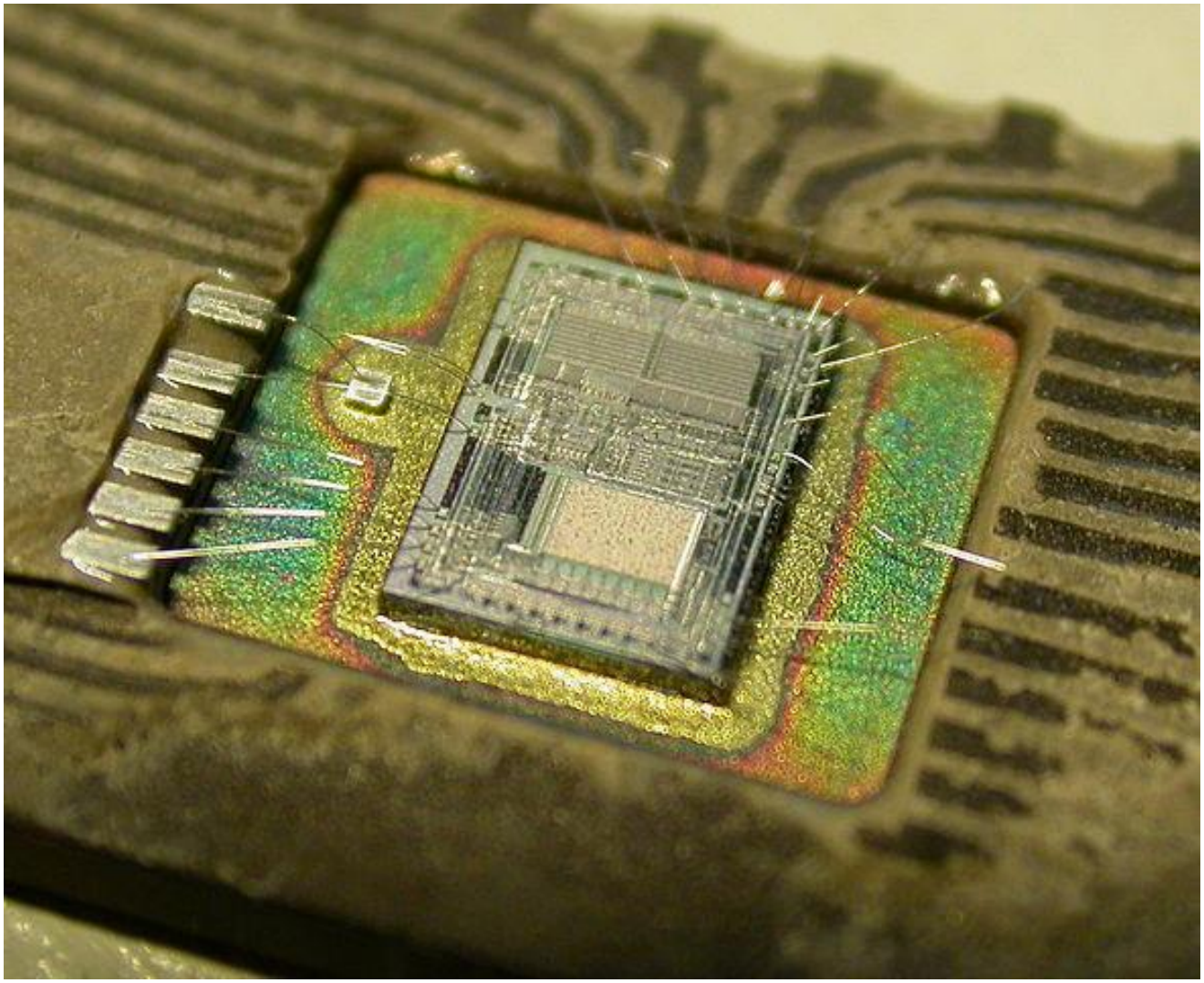

Рисунок 1.2 – Внутрішній вид мікроконтролеру «Intel 8051»

На фото (рис. 1.2) представлено відкритий кремнієвий кристал МК сімейства 8051. У центрі розміщена складна інтегральна схема, де знаходяться модулі арифметико-логічного пристрою процесора, масиви оперативної пам'яті RAM та постійної пам'яті ROM. Мікропровідники з'єднують внутрішні модулі процесора, пам'яті та периферії із зовнішніми металевими виводами корпусу для взаємодії з іншими компонентами електронних систем [3, 12].

Основною особливістю мікроконтролерних систем є тісний зв’язок програмного та апаратного забезпечення. Програмне забезпечення (ПЗ, firmware) відповідає за керування периферійними пристроями, обробку даних із сенсорів, реалізацію алгоритмів керування та забезпечення взаємодії із зовнішнім середовищем [4, 11].

Embedded-системи, побудовані на базі мікроконтролерів, працюють у режимі реального часу. Тобто, система повинна виконувати певні операції у визначені часові проміжки. Порушення часових обмежень може призводити до помилок у роботі пристрою або повної відмови системи. Тому для мікроконтролерних програм важливими є стабільність роботи, оптимізація продуктивності та контроль використання ресурсів, що є основним завданням розробників таких пристроїв.

Сучасні мікроконтролери поділяються на декілька основних архітектурних сімейств. Одними з найбільш поширених є ARM Cortex-M, AVR, PIC, ESP32 та STM32. Мікроконтролери сімейства AVR широко використовуються у навчальних та простих проєктах, зокрема в платформі Arduino. STM32 та ARM Cortex-M орієнтовані на більш продуктивні промислові та комерційні системи. ESP32 отримав значне поширення завдяки підтримці бездротових технологій Wi-Fi та Bluetooth, що робить його популярним рішенням для IoT-пристроїв [8]

Розвиток embedded-систем тісно пов’язаний із поширенням концепції Інтернету речей (Internet of Thing або IoT). IoT-пристрої використовують МК для збору, обробки та передачі інформації між фізичними об’єктами через мережу Інтернет. За оцінками міжнародних аналітичних компаній, кількість IoT-пристроїв у світі обчислюється десятками мільярдів одиниць, а ринок embedded-систем продовжує активно зростати. Це сприяє підвищенню вимог до якості ПЗ та необхідності використання сучасних методів тестування firmware [12, 15].

Особливістю ПЗ для МК є обмеженість апаратних ресурсів. Розробникам необхідно враховувати невеликий обсяг оперативної пам’яті, обмежену обчислювальну потужність та необхідність оптимізації енергоспоживання. Найбільшої актуальності для портативних пристроїв та автономних сенсорних систем є умова роботи від акумуляторів або батарей.

Ще однією важливою характеристикою embedded-систем є необхідність забезпечення високої надійності роботи. МК часто використовуються у критично важливих системах, де програмні помилки можуть призводити до небезпечних наслідків. Це стосується автомобільної електроніки, медичного обладнання, промислових систем керування та робототехнічних комплексів, різних сфер, напрямів та роду застосування. У зв'язку з цим важливого значення набувають процеси тестування та перевірки ПЗ мікроконтролерних систем [4,14].

Таким чином, МК є важливими спеціалізованими обчислювальними пристроями, які забезпечують функціонування сучасних embedded-систем. Їх широке застосування у різних сферах та постійне ускладнення ПЗ обумовлюють необхідність використання сучасних підходів до розробки, налагодження та автоматизованого тестування ПЗ.

1.1.2 Сфери застосування мікроконтролерних систем

Мікроконтролерні системи широко застосовуються у сучасних електронних та інформаційних технологіях. Завдяки компактності, низькому енергоспоживанню, відносно невисокій вартості та можливості виконання спеціалізованих задач у режимі реального часу, такі пристрої є основою великої кількості embedded-систем. Постійний розвиток мікроелектроніки та бездротових технологій сприяє подальшому поширенню МК у різних галузях промисловості та повсякденного життя.

Однією з найбільш перспективних сфер використання МК є IoT [9]. IoT-системи об'єднують фізичні пристрої у спільну мережу для збору, обробки та передачі даних через мережу Інтернет. МК в таких системах виконують функції обробки інформації із сенсорів, керування периферійними пристроями та реалізації бездротового зв'язку. IoT-пристрої використовуються у системах моніторингу навколишнього середовища, «розумних будинках», системах безпеки, логістиці та промисловій автоматизації (табл 1.1).

Таблиця 1.1 – Приклади МК, їх характеристики і сфери застосування

Назва МК	**Основні характеристики**	**Основні сфери застосування**
ATmega328P	8-бітний МК AVR, використовується у платформі Arduino Uno	Навчальні проєкти, робототехніка, прості embedded-системи
ESP8266	Підтримка Wi-Fi, низьке енергоспоживання	IoT-пристрої, smart home системи
ESP32	Підтримка Wi-Fi та Bluetooth, двоядерний процесор	IoT, бездротові системи, мультимедійні пристрої
STM32F103	ARM Cortex-M3, висока продуктивність, велика кількість периферії	Промислова автоматизація, робототехніка
Raspberry Pi Pico (RP2040)	Двоядерний ARM Cortex-M0+, підтримка MicroPython	Освітні та embedded-проєкти
PIC16F877A	Популярний мікроконтролер сімейства PIC	Системи керування та автоматизації

Наведені у таблиці МК відрізняються архітектурою, продуктивністю та набором периферійних модулів, однак усі вони широко застосовуються у сучасних embedded-системах.

Важливою сферою застосування таких рішень є автомобільна промисловість як і вся транспортна сфера. Сучасні транспортні засоби містять велику кількість електронних блоків керування, побудованих на базі МК. Вони відповідають за роботу двигуна, антиблокувальної системи гальмування, систем стабілізації, клімат-контролю, освітлення та мультимедійних комплексів. У сучасних автомобілях кількість embedded-пристроїв може досягати кількох десятків або сотень модулів керування. Надійність ПЗ таких систем має критичне значення, оскільки програмні помилки можуть впливати на безпеку експлуатації транспортного засобу [6]

Широкого застосування МК набули у робототехніці. Вони використовуються для керування електродвигунами, сервоприводами, сенсорами та системами навігації. Мікроконтролерні системи забезпечують виконання алгоритмів стабілізації, керування рухом роботів та обробки інформації від периферійних пристроїв. У робототехнічних системах важливими є швидкодія, точність виконання команд та здатність працювати у режимі реального часу.

Embedded-системи також активно використовуються у сфері розумних пристроїв, та розробки рішень розумних будинків. До таких систем належать «розумні» системи освітлення, побутова техніка, системи контролю доступу, електронні замки, smart-годинники та інші пристрої автоматизації побутових процесів. Основною перевагою таких систем є можливість автономної роботи та інтеграції із мобільними додатками або хмарними сервісами.

У промисловості МК використовуються в автоматизованих системах керування технологічними процесами. Вони забезпечують контроль параметрів обладнання, збір телеметричних даних, взаємодію із сенсорами та виконавчими механізмами.

У медичній сфері МК застосовуються у кардіомоніторах, системах контролю стану пацієнтів, портативних діагностичних пристроях, інсулінових помпах та іншому медичному обладнанні. Для таких систем особливо важливими є стабільність роботи, точність обробки даних та надійність ПЗ [6**Error! Reference source not found.**].

Значне поширення embedded-систем призводить до постійного ускладнення ПЗ мікроконтролерних пристроїв. У сучасних системах ПЗ повинно забезпечувати одночасну роботу великої кількості периферійних модулів, підтримку мережевих протоколів, обробку даних у режимі реального часу та взаємодію із зовнішніми сервісами. Це підвищує вимоги до якості розробки та тестування ПЗ.

Таким чином, мікроконтролерні системи є невід’ємною складовою сучасних електронних технологій та використовуються у великій кількості сфер діяльності людини. Їх широке застосування у критично важливих системах обумовлює

необхідність підвищення надійності ПЗ та використання сучасних методів автоматизованого тестування ПЗ.

1.1.3 Вимоги до надійності мікроконтролерних систем

Надійність є однією з ключових характеристик мікроконтролерних систем. На відміну від більшості настільних програм, помилки у ПЗ можуть призводити не лише до його некоректної роботи, а й до відмови обладнання, втрати даних або порушення роботи пов'язаних систем. Тому під час розробки мікроконтролерних пристроїв особлива увага приділяється забезпеченню стабільності та передбачуваності їх функціонування. У табл. 1.2 виділені основні вимоги до надійності мікроконтролерних систем [15].

Таблиця 1.2 – Основні вимоги до надійності мікроконтролерних систем

Вимога	Характеристика
Безвідмовність	Здатність системи виконувати свої функції протягом визначеного часу без помилок
Стабільність роботи	Коректне функціонування за різних режимів навантаження
Відмовостійкість	Збереження працездатності при виникненні окремих помилок
Відновлюваність	Можливість автоматичного відновлення після збоїв
Передбачуваність	Стабільність часових характеристик виконання програм
Безпечність	Відсутність дій, що можуть призвести до аварійних ситуацій

На надійність роботи впливають як апаратні, так і програмні фактори. До апаратних причин належать збої живлення, електромагнітні завади, помилки периферійних пристроїв та несправності окремих компонентів. Серед програмних причин найбільш поширеними є помилки алгоритмів, некоректна обробка

виняткових ситуацій, переповнення буферів, витоки пам’яті та помилки синхронізації [5, 14].

Особливу складність становить перевірка взаємодії ПЗ з периферійними пристроями. Навіть якщо окремі функції працюють коректно, помилки можуть виникати під час обміну даними через інтерфейси UART, SPI, I2C або при одночасній роботі кількох модулів системи. Саме тому перевірка лише окремих фрагментів коду не гарантує надійність роботи всієї системи.

Для підвищення надійності використовуються різні підходи: контроль помилок введення-виведення, механізми watchdog-таймерів, резервування критичних даних, діагностика стану системи та автоматичне відновлення після збоїв. Однак ефективність цих механізмів значною мірою залежить від якості тестування ПЗ на етапі розробки.

Зі збільшенням складності ПЗ кількість можливих сценаріїв роботи системи також зростає. Ручна перевірка всіх варіантів функціонування стає трудомісткою та не забезпечує достатнього рівня покриття тестами. У зв'язку з цим все більшого поширення набувають автоматизовані методи тестування, які дозволяють систематично перевіряти роботу програмного забезпечення та виявляти помилки ще до впровадження системи в експлуатацію [15].

Таким чином, забезпечення надійності є одним із головних завдань під час розробки мікроконтролерних систем. Високі вимоги до стабільності та безвідмовності роботи обумовлюють необхідність застосування сучасних методів контролю якості та автоматизованого тестування ПЗ.

1.2 Розробка і налагодження мікроконтролерних систем

Створення мікроконтролерної системи включає не лише розробку програмного коду, а й проєктування електричної схеми, налаштування периферійних модулів, перевірку взаємодії між компонентами та тестування готового рішення. Особливістю таких проєктів є необхідність одночасної роботи з апаратною та програмною частинами системи, що ускладнює процес розробки порівняно зі звичайними програмними продуктами.

У сучасній практиці розробки використовуються як фізичні налагоджувальні плати, так і програмні середовища моделювання, які дозволяють перевіряти роботу системи до виготовлення реального пристрою. Це скорочує час розробки, спрощує пошук помилок та знижує витрати на створення прототипів.

1.2.1 Етапи розробки апаратних і програмних засобів

Процес створення мікроконтролерної системи зазвичай складається з кількох взаємопов'язаних етапів. Узагальнену послідовність розробки наведено в табл. 1.3.

Таблиця 1.3 – Основні етапи розробки мікроконтролерних систем

Етап	Зміст робіт
Формування вимог	Визначення функцій майбутньої системи
Проєктування схеми	Вибір компонентів та розробка електричної схеми
Розробка ПЗ	Створення ПЗ для МК
Налагодження	Усунення помилок у схемі та програмі
Тестування	Перевірка коректності роботи системи
Впровадження	Використання готового рішення

Після визначення вимог до системи виконується вибір апаратної платформи та необхідних периферійних компонентів. На цьому етапі враховуються обчислювальні ресурси, кількість інтерфейсів, вимоги до енергоспоживання та можливості подальшого розширення системи.

Наступним етапом є створення ПЗ, яке реалізує необхідну функціональність. У процесі розробки виконуються налаштування периферійних модулів, реалізуються алгоритми обробки даних та забезпечується взаємодія між окремими компонентами системи.

Після завершення розробки виконується налагодження та тестування, під час яких виявляються та усуваються помилки, що виникають як у програмному коді, так і в апаратній частині системи.

1.2.2 Використання налагоджувальних плат і середовищ моделювання

Для пришвидшення розробки широко використовуються налагоджувальні плати, які дозволяють працювати з МК без створення власної друкованої плати. Такі рішення містять необхідні елементи живлення, програмування та підключення периферії [**Error! Reference source not found.**].

Серед найбільш поширених платформ можна виділити Arduino Uno, ESP32 DevKit, STM32 Nucleo та Raspberry Pi Pico. Вони використовуються як у навчальних цілях, так і під час створення прототипів реальних пристроїв.

Паралельно з фізичними платами використовуються середовища моделювання (табл. 1.4), які дозволяють перевіряти роботу схеми та програмного забезпечення у віртуальному середовищі. Основні переваги моделювання:

- відсутність необхідності у фізичному обладнанні;
- швидке внесення змін до схеми;
- можливість тестування великої кількості конфігурацій;
- спрощення навчального процесу.

Таблиця 1.4 – Популярні середовища моделювання мікроконтролерних систем

Середовище	**Основне призначення**
Proteus	Моделювання електронних схем та мікроконтролерів
Tinkercad	Навчальні Arduino-проєкти
SimulIDE	Просте моделювання електроніки
Wokwi	Симуляція Arduino, ESP32 та Raspberry Pi Pico у браузері

Особливий інтерес для даної роботи становить середовище Wokwi, яке дозволяє створювати схеми, завантажувати програмний код та перевіряти роботу проєкту без використання фізичного обладнання. Крім того, структура проєкту Wokwi зберігається у вигляді набору файлів, що відкриває можливість їх подальшого автоматизованого аналізу.

1.2.3 Тестування і налагодження мікроконтролерних систем

Налагодження є невід'ємною частиною процесу розробки. Його метою є пошук причин некоректної роботи системи та усунення виявлених помилок. Для цього використовуються монітори послідовного порту, логування, засоби покрокового виконання програм та аналіз роботи периферійних модулів.

Після завершення налагодження виконується тестування системи. Його основною метою є перевірка відповідності результатів роботи поставленим вимогам [19].

У загальному випадку можна виділити такі рівні тестування:

- модульне тестування;
- інтеграційне тестування;
- системне тестування;
- тестування взаємодії з периферійними пристроями.

Для невеликих проєктів перевірка часто виконується вручну шляхом запуску програми та аналізу отриманих результатів. Однак зі збільшенням складності системи та кількості тестових сценаріїв ефективність такого підходу знижується.

Особливо актуальною ця проблема є в навчальному процесі, де необхідно перевіряти велику кількість студентських робіт. У таких випадках перевірка програмного коду та електронної схеми займає значний час і залежить від досвіду викладача. Це створює передумови для використання автоматизованих засобів аналізу проєктів та оцінювання правильності їх виконання.

Таким чином, сучасні інструменти розробки дозволяють значно спростити створення та налагодження мікроконтролерних систем. Разом із тим зростання складності проєктів та необхідність перевірки великої кількості варіантів виконання завдань обумовлюють потребу у використанні автоматизованих методів тестування, які будуть розглянуті у наступному підрозділі.

1.3 Огляд існуючих засобів автоматизованого тестування

Складність сучасних мікроконтролерних проєктів призвела до появи великої кількості інструментів, призначених для автоматизації процесів перевірки ПЗ. Такі засоби дозволяють скоротити час тестування, підвищити якість програмного коду

та зменшити вплив людського фактора під час перевірки результатів розробки. Проте більшість існуючих рішень орієнтовані на окремі аспекти тестування і не забезпечують комплексної перевірки програмного коду та структури мікроконтролерного проєкту.

1.3.1 Засоби модульного тестування embedded-програм

Модульне тестування передбачає перевірку окремих функцій або програмних модулів незалежно від решти системи. Основною метою такого підходу є виявлення помилок на ранніх етапах розробки та спрощення подальшого супроводу ПЗ [**Error! Reference source not found.**].

Для мікроконтролерних систем найбільш поширеними засобами модульного тестування є Unity Test Framework, Ceedling, Google Test та вбудовані засоби PlatformIO (табл 1.5). Вони дозволяють автоматично запускати набір тестів та перевіряти правильність роботи окремих програмних компонентів.

Таблиця 1.5 – Поширені засоби модульного тестування

Засіб тестування	**Особливості**
Unity Test Framework	Легка бібліотека для тестування програм на мові C
Ceedling	Автоматизація тестування проєктів на основі Unity
Google Test	Популярний фреймворк для програм на C++
PlatformIO Unit Testing	Інтегроване тестування для embedded-проєктів

Перевагою модульного тестування є можливість швидкої перевірки окремих функцій та алгоритмів без необхідності запуску всієї системи. Однак такий підхід не дозволяє перевірити правильність побудови електронної схеми, налаштування периферійних модулів або відповідність проєкту вимогам конкретного завдання [**Error! Reference source not found.**].

1.3.2 Засоби моделювання та емуляції мікроконтролерних систем

Окремий напрямок розвитку становлять середовища моделювання та емуляції. Їх основним призначенням є перевірка роботи проєкту без використання фізичного обладнання. У табл. 1.6 представлені поширені рішення.

Таблиця 1.6 – Засоби моделювання мікроконтролерних систем [21, 22, 23]

Середовище	**Основні можливості**
Proteus	Моделювання схем та виконання програмного коду
Tinkercad	Навчальне середовище для Arduino-проєктів
SimulIDE	Спрощене моделювання електронних схем
Wokwi	Онлайн-симуляція мікроконтролерних систем

На відміну від засобів модульного тестування, системи моделювання дозволяють оцінити взаємодію програмного забезпечення з компонентами схеми. Розробник може перевіряти роботу датчиків, індикаторів, кнопок, модулів зв’язку та інших елементів ще до створення фізичного прототипу.

Особливістю Wokwi є представлення проєкту у вигляді набору файлів, які містять програмний код та опис електронної схеми. Така структура спрощує обмін проєктами та відкриває можливість автоматизованого аналізу їх вмісту сторонніми програмними засобами [**Error! Reference source not found.**].

Основним призначенням подібних середовищ залишається моделювання роботи системи. Вони не виконують автоматичну перевірку правильності реалізації поставленого завдання та не формують висновки щодо відповідності проєкту визначеним вимогам.

1.3.3 Підходи до автоматизованого аналізу коду та структури проєктів

Окрім класичного тестування, у сучасній розробці активно використовуються засоби автоматизованого аналізу програмного коду. Їх завданням є перевірка структури проєкту, пошук потенційних помилок та контроль дотримання визначених правил програмування [25].

До найбільш поширених підходів належать:

- статичний аналіз програмного коду;
- аналіз структури проєкту;
- перевірка відповідності програмного коду стандартам;
- аналіз залежностей між компонентами;
- автоматична перевірка конфігураційних файлів.

На відміну від модульного тестування, статичний аналіз не виконує програму безпосередньо, а досліджує її структуру. Це дозволяє виявляти помилки ще до запуску ПЗ.

Для проєктів, створених у середовищах моделювання, додатковий інтерес становить аналіз структури схеми. Наприклад, можна автоматично перевіряти наявність необхідних компонентів, правильність підключення окремих модулів або відповідність конфігурації вимогам завдання [24].

Такий підхід є особливо корисним у навчальному процесі, де важливо не лише перевірити працездатність програми, а й оцінити правильність реалізації проєкту в цілому [26].

1.3.4 Обґрунтування необхідності розробки власного рішення

Проведений аналіз показує, що існуючі засоби автоматизації вирішують окремі задачі процесу тестування. Фреймворки модульного тестування орієнтовані на перевірку програмного коду, а середовища моделювання забезпечують перевірку роботи схеми та ПЗ у віртуальному середовищі.

Проте відсутнє універсальне комплексне рішення, яке дозволяло б автоматично аналізувати готовий проєкт мікроконтролерної системи та визначати його відповідність поставленому завданню. У більшості випадків така перевірка виконується вручну шляхом перегляду програмного коду, аналізу схеми та запуску симуляції.

Особливо актуальною ця проблема є для навчального процесу, де викладачу необхідно перевіряти значну кількість однотипних лабораторних або практичних

робіт. Ручна перевірка займає багато часу, а результати можуть залежати від суб'єктивної оцінки.

Підходом до вирішення цієї проблеми є розробка системи автоматизованого тестування, здатної аналізувати структуру проєкту, програмний код та електронну схему, сформовані у середовищі моделювання. Використання проєктів Wokwi як вхідних даних дозволяє реалізувати таку перевірку без необхідності роботи з фізичним обладнанням.

В ході постановки завдання бакалаврської дипломної роботи виникла потреба у створенні програмного засобу, який забезпечить автоматизовану перевірку мікроконтролерних проєктів та формування результатів тестування на основі аналізу коду і структури схеми.

Метою даної роботи є підвищення ефективності тестування і налагодження мікрококонтролерних систем шляхом створення спеціалізованого програмного засобу автоматизованого тестування.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз сучасних мікроконтролерів та сфер їх застосування;
- дослідити існуючі методи розробки, налагодження та тестування програмного забезпечення embedded-систем
- виконати аналіз сучасних засобів автоматизованого тестування та моделювання мікроконтролерних проєктів
- сформулювати вимоги до системи автоматизованого тестування
- розробити структуру та архітектуру програмного забезпечення
- реалізувати прототип системи автоматизованого аналізу проєктів Wokwi.

Висновки до розділу 1

В першому розділі мікроконтролер проаналізовано як обчислювальний пристрій, визначені сфери застосування мікроконтролерів а також вимоги до надійності мікроконтролерних систем. Виконано аналіз процесу розробки і налагодження мікроконтролерних систем. Визначено роль налагоджувальних плат і середовищ моделювання у процесі розробки. Розглянуто необхідність

використання автоматизованих методів тестування мікроконтролерних систем. Проведено огляд існуючих засобів автоматизованого тестування. Визначені їх переваги і недоліки. Обґрунтовано необхідність створення спеціалізованого програмного засобу тестування. Сформульовано мету і задачі роботи.

2 ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

2.1 Особливості тестування ПЗ мікроконтролерних систем

2.1.1 Особливості програм для мікроконтролерів

Мікроконтролерні системи є основою значної кількості сучасних вбудованих пристроїв, що використовуються в промисловості, робототехніці, системах автоматизації, побутовій техніці та пристроях Інтернету речей. Особливістю таких систем є поєднання програмного забезпечення та апаратної частини в єдиному функціональному комплексі. На відміну від традиційного програмного забезпечення для персональних комп’ютерів, програми для МК безпосередньо взаємодіють з фізичними пристроями, здійснюючи керування датчиками, виконавчими механізмами та периферійними модулями.

Робота програми МК передбачає виконання низки спеціалізованих операцій, пов’язаних із налаштуванням цифрових та аналогових входів і виходів, обробкою сигналів від зовнішніх пристроїв, формуванням керуючих впливів, а також організацією обміну даними через комунікаційні інтерфейси. Коректність функціонування такої програми залежить не лише від правильності алгоритму, але й від відповідності програмного коду електричній схемі пристрою.

Однією з характерних особливостей програм для МК є наявність початкового етапу конфігурування апаратних ресурсів. На цьому етапі визначаються режими роботи портів введення-виведення, налаштовуються таймери, модулі переривань, аналого-цифрові перетворювачі та інші периферійні блоки. Помилки під час конфігурування можуть призвести до некоректної роботи всієї системи навіть за умови правильного виконання основного алгоритму.

Ще однією особливістю є необхідність забезпечення відповідності між програмним кодом та апаратними підключеннями. Наприклад, якщо світлодіод фізично підключений до певного цифрового виводу мікроконтролера, то саме цей вивід повинен використовуватись у програмі для керування пристроєм. Невідповідність між схемою та кодом є однією з найпоширеніших причин помилок під час виконання навчальних проєктів (рис. 2.1).

Рисунок 2.1 – Взаємодія ПЗ з апаратними компонентами мікроконтролерної системи

У процесі навчання студентів розробці мікроконтролерних систем значна кількість завдань пов’язана зі створенням простих схем керування світлодіодами, кнопками, звуковими сигналізаторами, датчиками та іншими периферійними пристроями. Перевірка правильності виконання таких робіт потребує аналізу як програмного коду, так і структури електронної схеми. Саме ця особливість робить доцільним використання спеціалізованих засобів автоматизованого тестування, здатних одночасно аналізувати декілька складових мікроконтролерного проєкту.

2.1.2 Контроль використання периферійних пристроїв

Однією з основних особливостей мікроконтролерних систем є їх взаємодія з периферійними пристроями. До таких пристроїв належать світлодіоди, кнопки, звукові сигналізатори, датчики, дисплеї та інші електронні компоненти, що підключаються до входів та виходів мікроконтролера. Коректність роботи всієї системи залежить від правильності як електричних підключень, так і програмного керування периферійними пристроями.

Під час розробки навчальних проєктів студенти часто допускають помилки, пов’язані з неправильним вибором виводів мікроконтролера, відсутністю необхідних компонентів у схемі або невідповідністю між програмним кодом та фактичним підключенням пристроїв. Наприклад, світлодіод може бути підключений до одного цифрового виводу, тоді як у програмі використовується інший. Аналогічні помилки можуть виникати під час роботи з кнопками, датчиками та іншими периферійними елементами.

Традиційний підхід до перевірки таких проєктів передбачає ручний аналіз електронної схеми та програмного коду викладачем. За великої кількості студентських робіт цей процес потребує значних витрат часу та підвищує ймовірність пропуску окремих помилок. Тому доцільним є застосування автоматизованих засобів контролю, здатних виконувати перевірку складу компонентів, правильності їх підключення та відповідності програмного коду визначеним вимогам.

У розроблюваній системі автоматизованого тестування контроль периферійних пристроїв реалізується шляхом аналізу структури електронної схеми проєкту та порівняння отриманих результатів із набором правил тестування. Такий підхід дозволяє автоматично виявляти типові помилки підключення та забезпечує об’єктивну оцінку правильності виконання навчального завдання.

2.1.3 Проблеми ручного тестування навчальних проєктів

У процесі вивчення дисциплін, пов’язаних із проєктуванням та програмуванням мікроконтролерних систем, студенти виконують значну кількість практичних і лабораторних робіт. Такі роботи, як правило, включають розробку

електронної схеми та ПЗ для реалізації визначеного функціоналу. Прикладами подібних завдань є керування світлодіодами, обробка сигналів від кнопок, робота з датчиками або створення простих автоматизованих систем керування.

Традиційна перевірка виконаних робіт здійснюється викладачем шляхом аналізу електронної схеми та програмного коду. Для цього необхідно перевірити наявність усіх необхідних компонентів, правильність їх підключення, відповідність програмного коду вимогам завдання та коректність використання периферійних пристроїв. За умови великої кількості студентських робіт такий підхід потребує значних витрат часу (рис. 2.2).

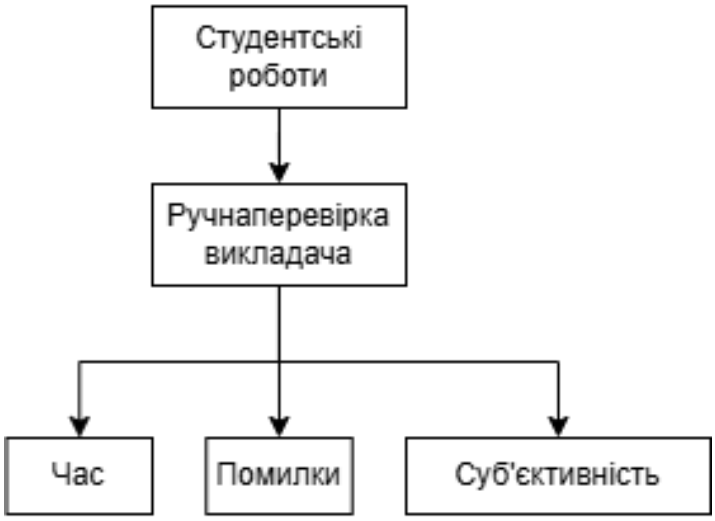

Рисунок 2.2 – Основні недоліки ручного тестування навчальних проєктів мікроконтролерних систем

Додатковою проблемою є можливість виникнення суб’єктивних помилок під час перевірки. Наприклад, викладач може не помітити відсутність окремого компонента, неправильне підключення пристрою або невідповідність між програмним кодом та електронною схемою. У результаті оцінювання різних робіт може виконуватись із неоднаковим рівнем деталізації.

Сучасні середовища моделювання, зокрема Wokwi, дозволяють створювати електронні схеми та програмний код у цифровому вигляді. Це створює передумови для автоматизації перевірки навчальних проєктів, оскільки структура схеми та

вихідний код можуть бути представлені у формалізованому вигляді та проаналізовані програмними засобами.

Одним із перспективних напрямів вирішення зазначеної проблеми є використання систем автоматизованого тестування, які виконують перевірку проєкту відповідно до заздалегідь визначених правил. Такий підхід дозволяє скоротити час перевірки, забезпечити єдині критерії оцінювання та підвищити об’єктивність контролю результатів навчання.

Таким чином, актуальною задачею є розробка програмного засобу, здатного автоматично аналізувати структуру мікроконтролерного проєкту, перевіряти відповідність схеми та програмного коду вимогам завдання, а також формувати звіт про результати перевірки.

2.1.4 Постановка задачі автоматизації перевірки

Аналіз особливостей програмного забезпечення мікроконтролерних систем та проблем ручного тестування навчальних проєктів показав доцільність застосування автоматизованих засобів контролю результатів виконання завдань. Використання таких засобів дозволяє зменшити навантаження на викладача, скоротити час перевірки робіт та забезпечити єдині критерії оцінювання.

Об’єктом автоматизованої перевірки в даній роботі є навчальні проєкти мікроконтролерних систем, створені у середовищі моделювання Wokwi. Результатом виконання такого проєкту є архів, що містить програмний код мікроконтролера та файл опису електронної схеми. Саме ці дані можуть бути використані для проведення автоматичного аналізу без необхідності запуску повної симуляції проєкту.

Основною задачею розроблюваної системи є перевірка відповідності проєкту вимогам конкретного навчального завдання. Для цього необхідно забезпечити автоматичний аналіз структури проєкту, перевірку наявності необхідних файлів, контроль складу електронної схеми, аналіз програмного коду та порівняння отриманих результатів із набором правил тестування.

Для забезпечення універсальності системи вимоги до конкретних завдань доцільно зберігати у вигляді окремих файлів правил. Такий підхід дозволяє використовувати одну систему для перевірки різних лабораторних та практичних робіт без зміни програмного коду програмного засобу.

У результаті роботи система повинна формувати звіт, який містить перелік виконаних перевірок, інформацію про виявлені помилки, оцінку рівня виконання завдання та загальний висновок щодо правильності створеного проєкту (рис. 2.3).

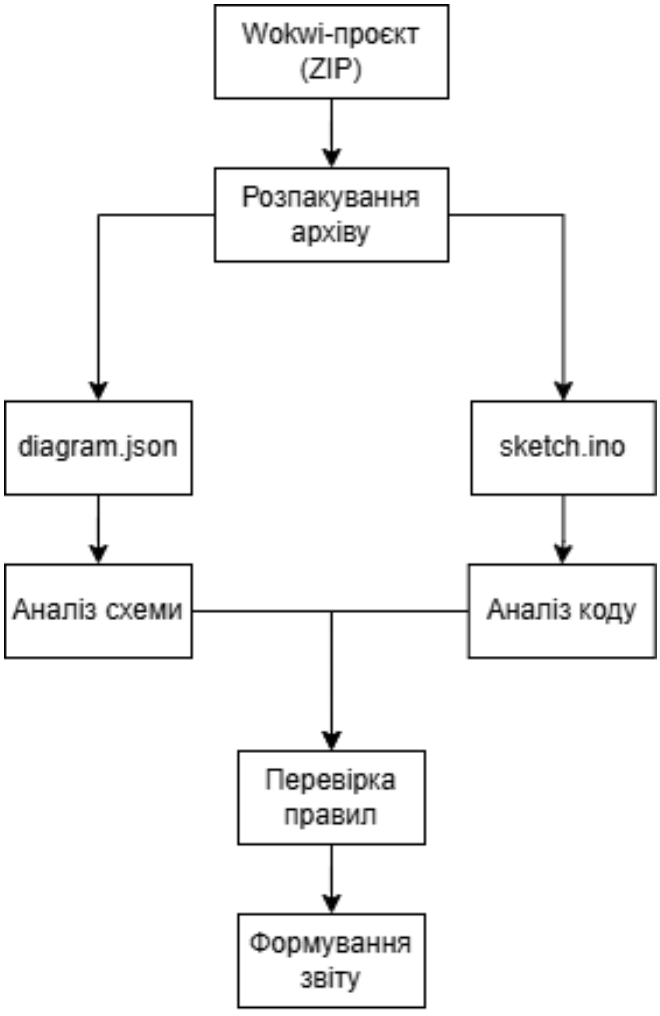

Рисунок 2.3 – Загальна концепція автоматизованої перевірки мікроконтролерних проєктів

Таким чином, необхідно розробити систему автоматизованого тестування програмного забезпечення мікроконтролерних систем, що забезпечує аналіз

проєктів середовища Wokwi на основі заданих правил перевірки та формує звіт про результати тестування.

2.2. Вимоги до системи автоматизованого тестування

2.2.1. Функціональні вимоги

Під час проєктування системи автоматизованого тестування програмного забезпечення мікроконтролерних систем було сформовано перелік функціональних вимог, які визначають основні можливості майбутнього програмного засобу (табл. 2.1). Враховуючи специфіку навчальних проєктів, розроблених у середовищі моделювання Wokwi, система повинна забезпечувати автоматичний аналіз як програмного коду, так і структури електронної схеми.

Таблиця 2.1 – Основні функціональні вимоги

Позначення	Вимога
F1	Завантаження Wokwi-проєкту у форматі ZIP
F2	Пошук та перевірка файлів проєкту
F3	Аналіз структури електронної схеми
F4	Аналіз програмного коду
F5	Перевірка відповідності правилам тестування
F6	Формування оцінки виконання завдання
F7	Генерування текстового звіту
F8	Генерування HTML-звіту
F9	Підтримка різних наборів правил тестування

Основною функцією системи є завантаження проєкту, підготовленого студентом у середовищі Wokwi. Для забезпечення зручності використання вхідними даними обрано ZIP-архів, який містить програмний код МК та файл опису електронної схеми.

Після завантаження проєкту система повинна автоматично виконувати пошук необхідних файлів, перевіряти їх наявність та здійснювати аналіз їх вмісту. Для

програмного коду повинна виконуватися перевірка наявності обов’язкових функцій та конструкцій, передбачених умовами завдання. Для електронної схеми необхідно забезпечити перевірку складу компонентів, правильності їх підключення та відповідності визначеним правилам тестування.

Система повинна підтримувати використання окремих файлів правил перевірки. Такий підхід дозволяє використовувати один програмний засіб для контролю різних лабораторних та практичних робіт без внесення змін до програмного коду самої системи. У файлах правил задаються перелік необхідних компонентів, обов’язкові елементи програмного коду та параметри підключення периферійних пристроїв.

За результатами аналізу система повинна формувати звіт, який містить перелік виконаних перевірок, їх результати, виявлені помилки та загальну оцінку правильності виконання завдання. Передбачено можливість збереження результатів перевірки у текстовому та HTML-форматах для подальшого використання викладачем або студентом.

2.2.2. Нефункціональні вимоги

Окрім функціональних можливостей, система автоматизованого тестування повинна відповідати ряду нефункціональних вимог, які визначають її зручність використання, ефективність роботи та можливість подальшого розвитку (табл. 2.2).

Таблиця 2.2 – Нефункціональні вимоги до системи

Позначення	Вимога
NF1	Простота використання
NF2	Можливість розширення наборів правил
NF3	Швидке виконання перевірки
NF4	Переносимість програмного забезпечення
NF5	Зручне представлення результатів
NF6	Модульна структура програмного забезпечення

Однією з основних вимог є простота використання. Система повинна мати зрозумілий графічний інтерфейс, який дозволяє користувачу завантажити проєкт для перевірки, обрати набір правил тестування та отримати результати аналізу без необхідності виконання додаткових налаштувань.

Важливою вимогою є незалежність від конкретного типу завдання. Система повинна підтримувати використання зовнішніх файлів правил, що дозволяє застосовувати її для перевірки різних навчальних проєктів без модифікації програмного коду. Такий підхід забезпечує можливість розширення функціональності шляхом створення нових наборів правил тестування.

Система повинна забезпечувати достатню швидкодію під час аналізу проєктів. Оскільки перевірка виконується шляхом аналізу текстових файлів та структури електронної схеми, час обробки одного проєкту повинен становити лише декілька секунд, що дозволяє використовувати програмний засіб для перевірки великої кількості студентських робіт.

Ще однією вимогою є переносимість програмного забезпечення. Використання мови програмування Python та бібліотеки PyQt6 дозволяє запускати систему на різних операційних системах без суттєвих змін у програмному коді.

Також система повинна забезпечувати наочне представлення результатів перевірки. Для цього передбачено формування звітів у текстовому та HTML-форматах із відображенням результатів окремих перевірок, підсумкової оцінки та загального висновку щодо виконання завдання.

2.2.3. Формування правил тестування

Однією з основних вимог до розроблюваної системи є можливість перевірки різних навчальних завдань без внесення змін до програмного коду. Для реалізації такої можливості було прийнято рішення використовувати окремі файли правил тестування, які визначають критерії оцінювання конкретного проєкту.

Правила тестування містять перелік компонентів, які повинні бути присутніми в електронній схемі, обов’язкові елементи програмного коду, а також параметри підключення периферійних пристроїв. Під час перевірки система завантажує

відповідний файл правил та порівнює отримані результати аналізу проєкту із заданими вимогами.

Такий підхід дозволяє використовувати одну систему для перевірки різних типів навчальних завдань. Наприклад, для завдання зі світлодіодом можуть перевірятися наявність світлодіода, резистора та використання функції digitalWrite(), тоді як для завдання з кнопкою додатково контролюється наявність компонента кнопки та використання функції digitalRead().

Для зберігання правил було обрано формат JSON. Даний формат має просту структуру, легко обробляється мовою програмування Python та забезпечує зручне редагування файлів без необхідності зміни програмного коду системи. Крім того, використання JSON дозволяє зберігати структуровані дані у вигляді пар «ключ – значення», що спрощує їх подальшу обробку.

У розробленому прототипі для кожного навчального завдання створюється окремий файл правил (рис. 2.4). Під час запуску перевірки користувач обирає не лише архів проєкту, але й файл правил, що відповідає конкретному завданню. Це забезпечує універсальність програмного засобу та спрощує його адаптацію до нових лабораторних і практичних робіт.

Рисунок 2.4 – Приклад структури файлу правил тестування

2.2.4. Формування звітів

Для забезпечення зручності аналізу результатів звіт повинен містити перелік усіх виконаних перевірок із зазначенням їх статусу. Для кожної перевірки необхідно відображати її назву, результат виконання та пояснення отриманого результату. Такий підхід дозволяє швидко визначити причину виникнення помилки та спрощує процес виправлення недоліків у проєкті.

Важливою вимогою є формування підсумкової оцінки виконання завдання. Для цього система повинна підраховувати кількість успішно виконаних перевірок та кількість виявлених помилок. На основі отриманих результатів обчислюється відсоток виконання завдання та формується загальний висновок щодо якості розробленого проєкту.

Для забезпечення можливості подальшого використання результатів тестування передбачено підтримку декількох форматів звітів. Текстовий формат забезпечує просте збереження та документування результатів перевірки, тоді як

HTML-звіт дозволяє відображати інформацію у структурованому вигляді з використанням кольорового оформлення та елементів форматування.

Звіт повинен бути зрозумілим як для викладача, так і для студента. Викладач отримує можливість швидко оцінити правильність виконання завдання, а студент може використати отриману інформацію для пошуку та усунення виявлених помилок.

Структура сформованого звіту повинна включати такі основні елементи:

- інформацію про виконані перевірки;
- статус кожної перевірки;
- пояснення результатів перевірки;
- кількість успішних перевірок;
- кількість виявлених помилок;
- підсумкову оцінку виконання завдання;
- загальний висновок щодо результатів тестування.

2.3. Проєктування структури програмного забезпечення

2.3.1. Вибір мови програмування і засобів розробки

Для розробки прототипу системи автоматизованого тестування було обрано мову програмування Python. Даний вибір обумовлений простотою реалізації програм обробки даних, наявністю вбудованих засобів роботи з JSON-файлами та ZIP-архівами, а також широким набором бібліотек для створення прикладного програмного забезпечення.

Для реалізації графічного інтерфейсу користувача використано бібліотеку PyQt6, яка забезпечує створення сучасних настільних додатків та підтримує основні елементи взаємодії з користувачем, необхідні для завантаження проєктів, запуску перевірки та відображення результатів тестування.

Як середовище розробки та моделювання мікроконтролерних систем було обрано платформу Wokwi. Дане середовище дозволяє створювати електронні схеми та програмний код у цифровому вигляді, а також експортувати проєкти у форматі, придатному для автоматизованого аналізу.

Обраний набір програмних засобів (табл. 2.3) забезпечує можливість реалізації всіх функцій системи автоматизованого тестування та дозволяє виконувати її подальше розширення без суттєвих змін архітектури програмного забезпечення.

Таблиця 2.3 – Програмні засоби

Засіб	**Призначення**
Python	Реалізація логіки системи
PyQt6	Графічний інтерфейс користувача
JSON	Зберігання правил тестування
Wokwi	Створення та моделювання проєктів
HTML	Формування звітів

2.3.2. Формати вхідних даних для тестування

Для проведення автоматизованого тестування система використовує дані проєкту, сформованого у середовищі моделювання Wokwi. Вхідними даними є ZIP-архів, який містить файли, необхідні для аналізу структури електронної схеми та програмного коду мікроконтролера.

Основними файлами проєкту є файл програмного коду з розширенням .ino та файл опису електронної схеми diagram.json. Файл програмного коду містить реалізацію алгоритму роботи мікроконтролерної системи, включаючи конфігурацію периферійних пристроїв та логіку їх взаємодії. Файл diagram.json містить структурований опис електронної схеми, перелік використаних компонентів та інформацію про їх з’єднання.

Крім архіву проєкту, для виконання перевірки використовується файл правил тестування у форматі JSON. У цьому файлі зберігаються вимоги до конкретного завдання, зокрема перелік обов’язкових компонентів, необхідні елементи програмного коду та параметри підключення периферійних пристроїв.

Під час запуску перевірки користувач обирає ZIP-архів проєкту та відповідний файл правил. Після цього система виконує розпакування архіву, аналіз його вмісту та порівняння отриманих даних із заданими вимогами (рис. 2.5).

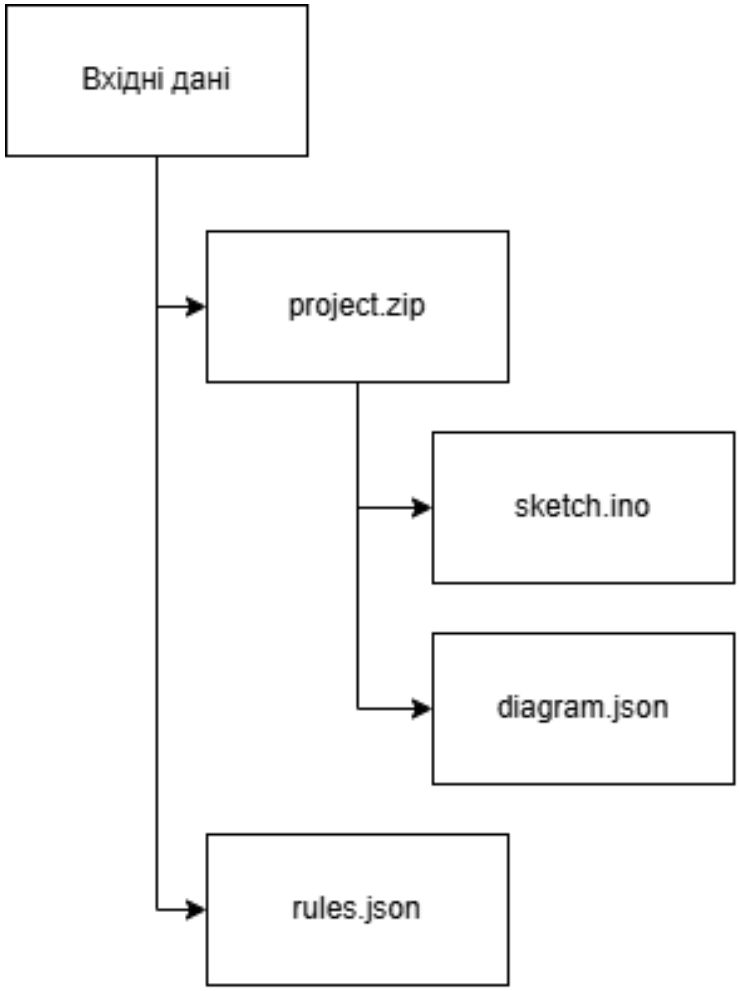

Рисунок 2.5 – Структура вхідних даних системи автоматизованого тестування

2.3.3. Формати вихідних файлів результатів тестування

Результатом роботи системи автоматизованого тестування є звіт про виконані перевірки. Формування звітів виконується після завершення аналізу структури проєкту, електронної схеми та програмного коду.

У розробленому прототипі реалізовано підтримку двох форматів звітів: текстового (TXT) та HTML. Текстовий формат використовується для збереження результатів перевірки у простому вигляді та може бути відкритий будь-яким текстовим редактором. Такий звіт містить перелік виконаних перевірок, їх результати та загальний висновок щодо правильності виконання завдання.

Для підвищення наочності також реалізовано формування HTML-звіту. У даному форматі результати перевірки відображаються у вигляді таблиці із кольоровим виділенням успішних та неуспішних перевірок. Крім того, HTML-звіт містить статистичну інформацію про кількість успішно виконаних перевірок, кількість помилок та підсумкову оцінку проєкту.

Використання двох форматів звітів забезпечує зручність роботи як для студентів, так і для викладачів. Текстовий звіт може використовуватись для документування результатів перевірки, а HTML-звіт забезпечує більш наочне представлення отриманих результатів.

2.3.4. Структура програмного забезпечення

Для забезпечення зручності розробки, супроводу та подальшого розширення функціональності система автоматизованого тестування побудована за модульним принципом (рис. 2.6). Кожен модуль відповідає за виконання окремого етапу обробки даних, що дозволяє спростити структуру програмного забезпечення та підвищити його гнучкість.

До складу системи входять модуль графічного інтерфейсу користувача, модуль аналізу проєкту, модуль перевірки відповідності правилам тестування та модуль формування звітів. Взаємодія між модулями здійснюється шляхом передачі результатів аналізу між окремими компонентами системи.

Рисунок 2.6 – Структура програмного забезпечення системи автоматизованого тестування

Модуль графічного інтерфейсу забезпечує взаємодію користувача із системою. За його допомогою виконується вибір архіву проєкту, завантаження файлу правил тестування, запуск перевірки та збереження сформованого звіту.

Модуль аналізу проєкту відповідає за розпакування архіву, пошук необхідних файлів та отримання інформації про структуру електронної схеми і програмний код. На цьому етапі виконуються перевірки складу компонентів, аналіз з’єднань між ними та пошук необхідних елементів програмного коду.

Модуль перевірки відповідності правилам виконує порівняння результатів аналізу з вимогами, визначеними у файлі правил тестування. Саме на цьому етапі формується перелік успішних перевірок та виявлених помилок.

Модуль формування звітів призначений для створення текстових та HTML-звітів, які містять результати перевірки та підсумкову оцінку виконання завдання.

Висновки до розділу 2

У другому розділі розглянуті особливості тестування програмного забезпечення мікроконтролерів, зокрема, пов’язані з необхідністю тестування периферійних пристроїв. Визначені проблеми, що виникають при ручному тестуванні навчальних проєктів мікроконтролерних систем. Виконано постановку задачі автоматизованого тестування. Сформульовані вимоги до системи автоматизованого тестування, що включають функціональні і нефункціональні вимоги. Сформульовані правила тестування ПЗ МК і формування звітів. Виконано проєктування структури програмного забезпечення. Визначені формати файлів вхідних даних і вихідних звітів. Створено структурну схему програмного забезпечення системи автоматизованого тестування.

3 РОЗРОБКА ПРОТОТИПУ СИСТЕМИ І ЙОГО ДОСЛІДЖЕННЯ

3.1. Розробка програмного забезпечення

Відповідно до сформованих вимог було розроблено прототип системи автоматизованого тестування програмного забезпечення мікроконтролерних систем. Основною задачею програмного засобу є аналіз навчальних проєктів, створених у середовищі моделювання Wokwi, та автоматичне визначення їх відповідності вимогам конкретного завдання.

Розробка виконувалась мовою програмування Python із використанням бібліотеки PyQt6 для реалізації графічного інтерфейсу користувача. Такий підхід дозволив створити автономний настільний застосунок, який не потребує додаткових серверних компонентів та може використовуватись безпосередньо на комп’ютері викладача або студента.

Структура програмного забезпечення побудована за модульним принципом. Основні функції системи розподілені між окремими файлами, що відповідають за графічний інтерфейс, виконання перевірок та формування звітів. Загальну структуру проєкту наведено на рис. 3.1, а призначення файлів – у табл. 3.1.

Таблиця 3.1 – Призначення основних файлів проєкту

Файл	**Призначення**
gui.py	графічний інтерфейс
checker.py	модуль перевірок
report.py	формування звітів
rules/*.json	правила тестування
examples/*	тестові проєкти

Файл gui.py реалізує графічний інтерфейс користувача та забезпечує вибір архіву проєкту, завантаження файлу правил тестування, запуск перевірки та збереження сформованих звітів. Основні перевірки реалізовані у модулі checker.py, який виконує аналіз структури проєкту, програмного коду та електронної схеми. Формування текстових та HTML-звітів здійснюється модулем report.py.

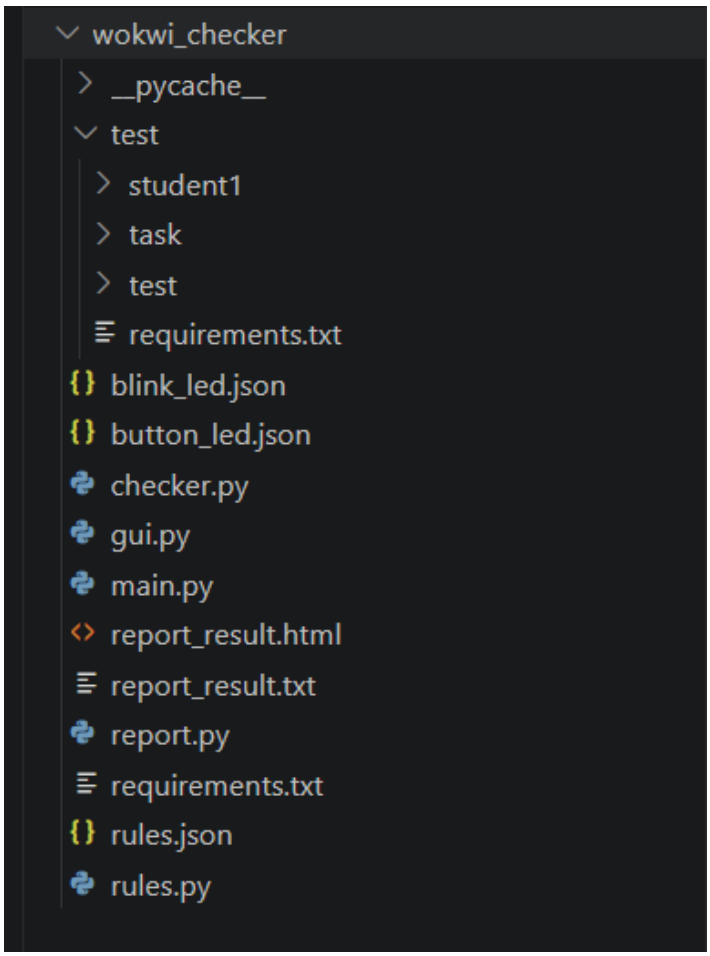

Рисунок 3.1 – Структура директорій проєкту

Робота системи починається із завантаження ZIP-архіву проєкту та відповідного файлу правил тестування. Після цього архів автоматично розпаковується у тимчасовий каталог, виконується пошук файлів програмного коду та опису електронної схеми. Отримані дані використовуються для подальшого аналізу.

Під час аналізу схеми система визначає перелік використаних компонентів та перевіряє правильність їх підключення. Для цього з файлу diagram.json зчитується інформація про компоненти та їх електричні з'єднання. На основі отриманих даних виконується пошук необхідних елементів схеми та перевірка відповідності підключень вимогам завдання.

Аналіз програмного коду здійснюється шляхом пошуку необхідних функцій та конструкцій мови Arduino. Зокрема перевіряється наявність функцій setup() та

loop(), а також використання функцій керування периферійними пристроями, визначених правилами тестування.

Для забезпечення універсальності системи використовується механізм зовнішніх файлів правил у форматі JSON. У таких файлах зберігається перелік необхідних компонентів, елементів програмного коду та параметрів підключення периферійних пристроїв. Це дозволяє використовувати одну систему для перевірки різних лабораторних і практичних робіт без зміни програмного коду.

За результатами перевірки формується набір повідомлень про успішні та неуспішні перевірки. На основі отриманих результатів розраховується відсоток виконання завдання та формується загальний висновок щодо правильності розробленого проєкту. Користувач може зберегти результати у вигляді текстового або HTML-звіту для подальшого використання.

Розроблений прототип реалізує основні функції автоматизованого тестування навчальних мікроконтролерних проєктів та створює основу для подальшого розширення системи шляхом додавання нових типів перевірок і наборів правил тестування.

3.2. Дослідження роботи системи

3.2.1. Формування набору тестових проєктів

Для перевірки працездатності розробленого прототипу було підготовлено набір тестових проєктів у середовищі Wokwi (рис. 3.2). Кожен проєкт відповідає окремому навчальному завданню та містить електронну схему і програмний код мікроконтролера Arduino Uno (табл. 3.2).

Метою тестування є перевірка коректності роботи алгоритмів аналізу структури проєкту, електронної схеми та програмного коду, а також оцінка здатності системи виявляти типові помилки під час виконання навчальних завдань.

Для проведення дослідження було підготовлено п'ять тестових проєктів, які охоплюють найбільш поширені сценарії використання периферійних пристроїв у навчальних роботах.

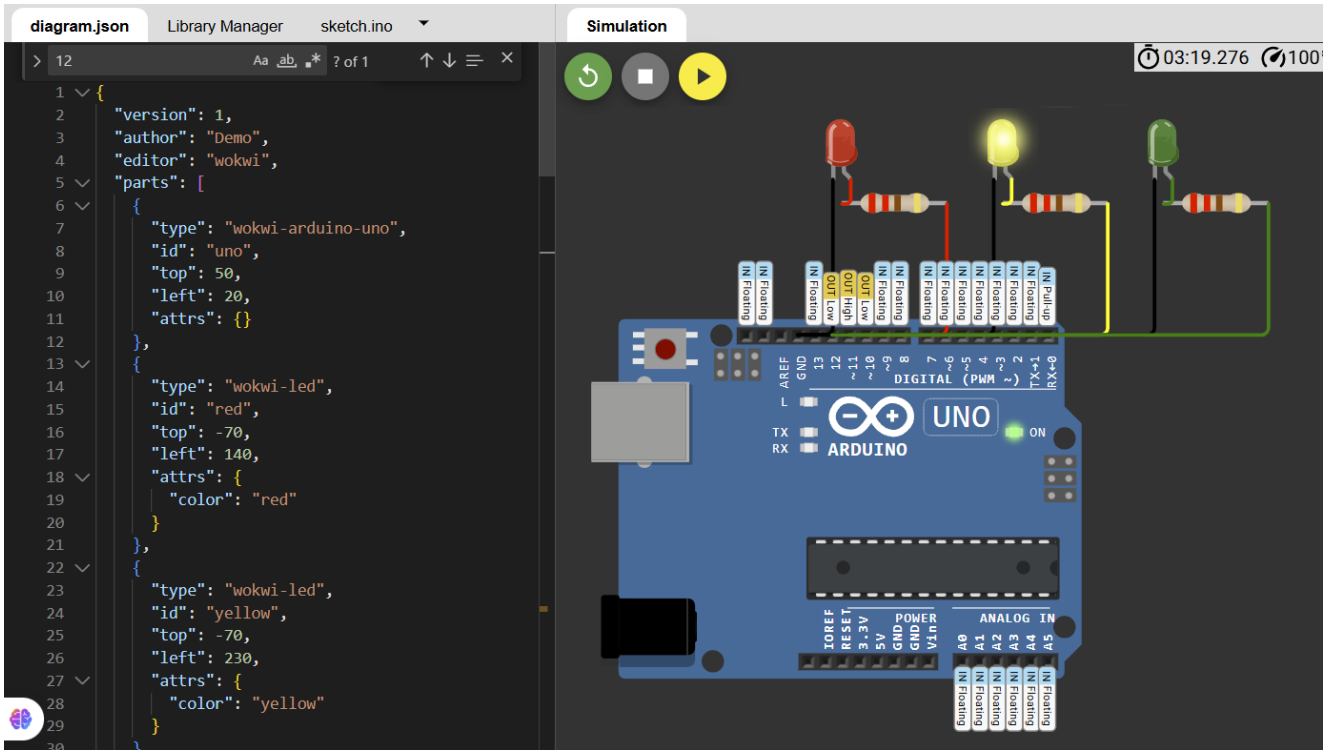

Рисунок 3.2 – Середовище моделювання Wokwi

Таблиця 3.2 – Опис тестових завдань

№	Назва завдання	Компоненти	Основні перевірки
1	Blink LED	Arduino, LED, резистор	Підключення LED, digitalWrite()
2	Button Controls LED	Arduino, LED, кнопка	digitalRead(), digitalWrite()
3	Buzzer Alarm	Arduino, кнопка, buzzer	tone(), digitalRead()
4	Potentiometer LED PWM	Arduino, потенціометр, LED	analogRead(), analogWrite()
5	Traffic Light	Arduino, 3 LED, 3 резистори	Багатокомпонентна схема

Усі проєкти були реалізовані в середовищі моделювання Wokwi та збережені у вигляді ZIP-архівів, що містять програмний код та опис електронної схеми.

Перше завдання передбачає реалізацію керування світлодіодом за допомогою мікроконтролера Arduino Uno. Студент повинен створити схему зі світлодіодом та резистором, підключити світлодіод до цифрового виводу D12 та реалізувати його періодичне блимання за допомогою функції digitalWrite(). Дане завдання використовується для перевірки базових можливостей аналізу схеми та програмного коду.

Друге завдання передбачає використання кнопки для керування світлодіодом. У схемі повинні бути присутні налагоджувальна плата Arduino Uno, світлодіод, резистор та кнопка. Система перевіряє наявність необхідних компонентів, правильність підключення кнопки та використання у програмному коді функції digitalRead() для зчитування її стану.

Третє завдання моделює роботу звукової сигналізації. Проєкт містить кнопку та звуковий модуль (buzzer), а програмний код повинен забезпечувати формування звукового сигналу при натисканні кнопки. Метою перевірки є контроль правильності використання периферійних пристроїв та відповідних функцій керування звуковим модулем.

Четверте завдання (рис. 3.3) присвячене роботі з аналоговими сигналами. У схемі використовується потенціометр, значення якого зчитується за допомогою аналогового входу мікроконтролера. Отримані дані застосовуються для керування яскравістю світлодіода через ШІМ-вихід. Під час тестування перевіряється використання функцій analogRead() та analogWrite(), а також відповідність підключення компонентів вимогам завдання (рис. 3.4).

П’яте завдання (рис. 3.5) є найбільш складним серед підготовлених тестових прикладів та реалізує модель світлофора. Схема містить три світлодіоди різного кольору та відповідні резистори. Програмний код повинен забезпечувати послідовне перемикання сигналів світлофора відповідно до визначеного алгоритму роботи. Для даного завдання система виконує перевірку кількості компонентів, правильності їх підключення та наявності необхідних програмних конструкцій.

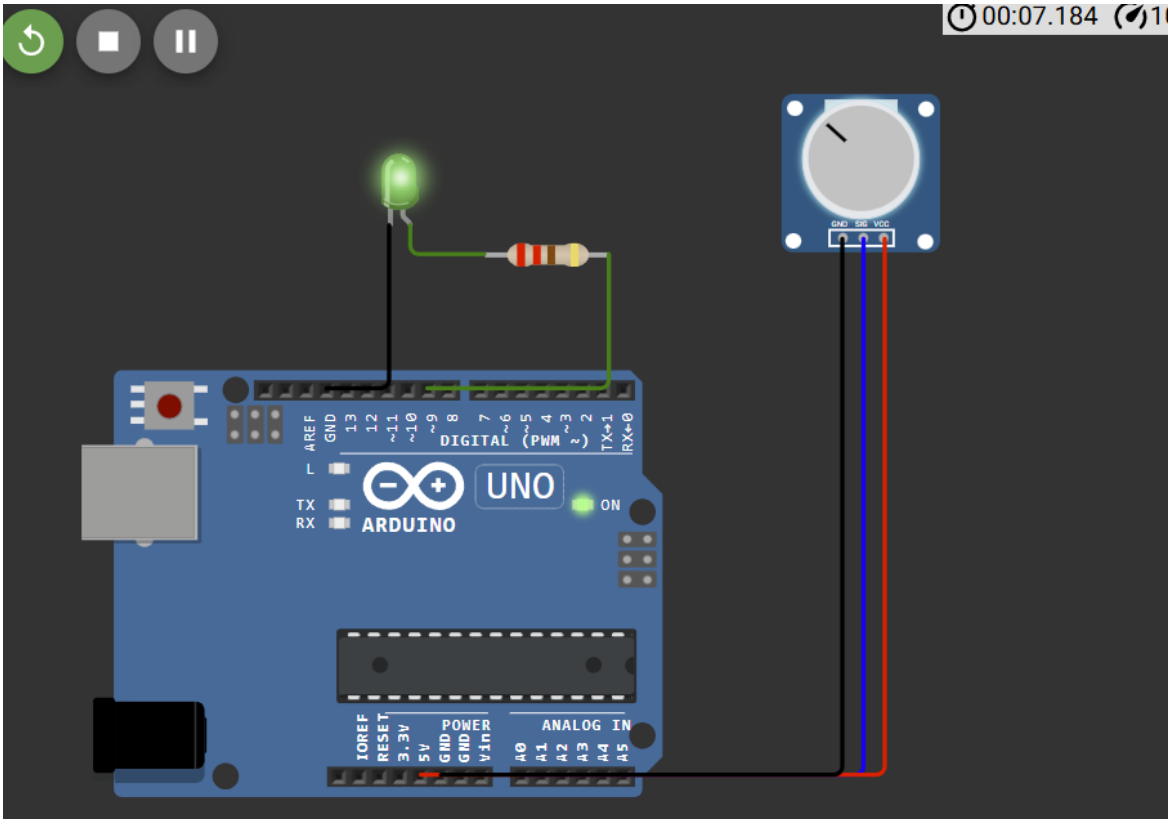

Рисунок 3.3 – Приклад виконання 4-го завдання «Potentiometer LED PWM»

main.py checker.py 05_traffic_light.json

wokwi_checker > test > task > wokwi_rules_and_tasks > rules > 05_traffic_light.json > ...

```
{
  "task_name": "Traffic light",
  "required_components": [
    "wokwi-arduino-uno",
    "wokwi-led",
    "wokwi-resistor"
  ],
  "required_code_elements": [
    "setup",
    "loop",
    "pinMode",
    "digitalWrite",
    "delay"
  ],
  "expected_led_pins": {
    "red": "10",
    "yellow": "11",
    "green": "12"
  },
  "minimum_led_count": 3,
  "minimum_resistor_count": 3
}
```

Рисунок 3.4 – Приклад завдання та опису правил одного з завдань

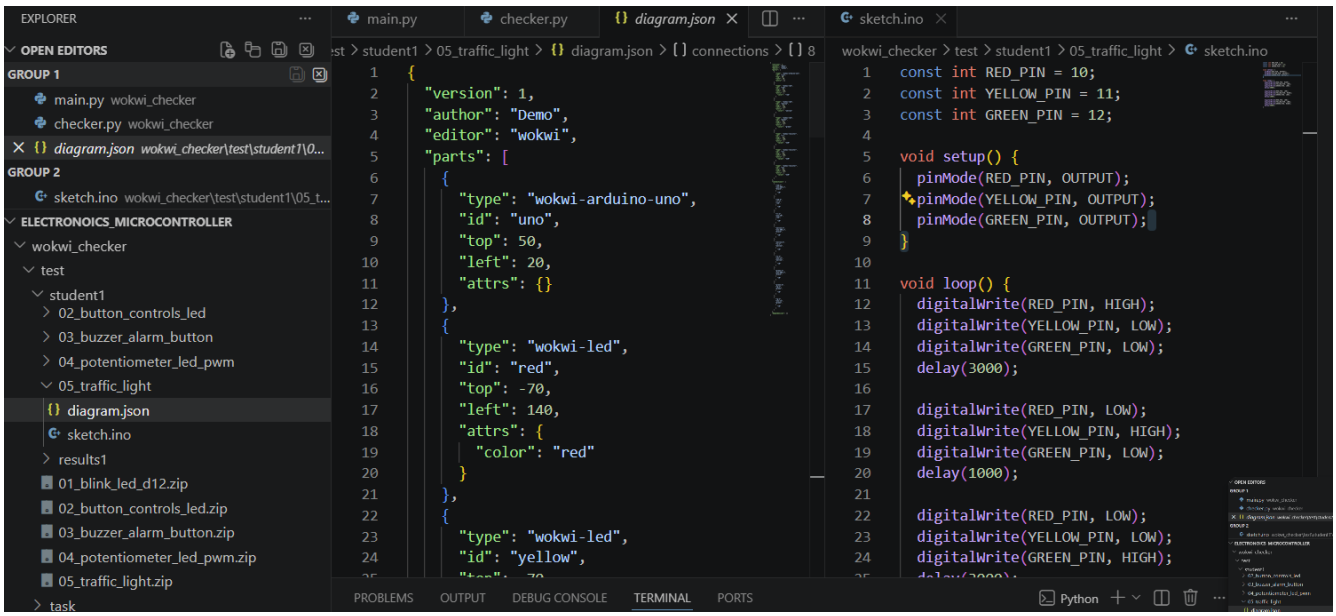

Рисунок 3.5 – Приклад розпакованого файлу ZIP, що містить виконане завдання, яке й буде перевірятися програмно «Traffic Light»

3.2.2. Порядок виконання тестування

Перед початком перевірки користувач запускає програму та обирає ZIP-архів проєкту і відповідний файл правил тестування. Після натискання кнопки запуску система автоматично розпочинає аналіз структури проєкту. Послідовність роботи користувача полягає у виборі архіву проєкту, завантаженні файлу правил, запуску перевірки та збереженні отриманого звіту.

Після початку перевірки програма виконує розпакування архіву та пошук необхідних файлів проєкту. На першому етапі здійснюється перевірка наявності файлу програмного коду та файлу опису електронної схеми. За відсутності одного з необхідних файлів подальша перевірка припиняється та користувачу виводиться відповідне повідомлення.

Наступним етапом є аналіз структури електронної схеми. Система визначає перелік використаних компонентів, аналізує їх з’єднання та перевіряє відповідність вимогам, зазначеним у файлі правил. Паралельно виконується аналіз програмного коду, під час якого здійснюється пошук необхідних функцій та програмних конструкцій.

Результати окремих перевірок відображаються у вікні програми у вигляді повідомлень про успішне або неуспішне виконання відповідних умов. Для кожної перевірки формується html-сторінка результату, на якій виводиться її назва, статус та коротке пояснення отриманого результату.

Після завершення аналізу користувач отримує підсумковий звіт у вікні програми (рис. 3.6). У звіті відображаються результати перевірки структури проєкту, складу електронної схеми, правильності підключення компонентів та аналізу програмного коду. Додатково визначається кількість успішних перевірок, кількість помилок та розраховується підсумкова оцінка виконання завдання.

Рисунок 3.6 – Відображення результатів у GUI

У випадку виявлення помилок система виводить повідомлення із зазначенням причини невідповідності. Це дозволяє користувачу швидко визначити помилковий елемент схеми або програмного коду та виконати необхідні виправлення.

Після завершення перевірки результати можуть бути збережені у текстовому форматі (рис. 3.7) або у вигляді HTML-файлу.

main.py checker.py report_result5.txt

wokwi_checker > test > student1 > results1 > report_result5.txt

```
ЗВІТ ПЕРЕВІРКИ WOKWI-ПРОЄКТУ
========================================

Файл .ino: OK — Файл програмного коду знайдено
Файл diagram.json: OK — Файл схеми знайдено
wokwi-arduino-uno: OK — Компонент знайдено
wokwi-led: OK — Компонент знайдено
wokwi-resistor: OK — Компонент знайдено
LED: OK — Світлодіод знайдено
Підключення LED: OK — LED green підключено до піна 12 через резистор rg
Відповідність коду і схеми: ERROR — У коді не знайдено використання піна 12
Код: setup: OK — Елемент setup знайдено
Код: loop: OK — Елемент loop знайдено
Код: pinMode: OK — Елемент pinMode знайдено
Код: digitalWrite: OK — Елемент digitalWrite знайдено
Код: delay: OK — Елемент delay знайдено

ПІДСУМОК
----------------------------------------
Успішних перевірок: 12
Помилок: 1
Загальний бал: 92.31%
Загальний висновок: проєкт потребує виправлення.

```

Рисунок 3.7 – Результат перевірки 5-го завдання у вигляді текстового репорту

Формування HTML-звіту виконується автоматично на основі отриманих результатів тестування (рис. 3.8). У сформованому звіті інформація відображається у вигляді структурованої таблиці з виділенням успішних та неуспішних перевірок.

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Звіт перевірки Wokwi-проєкту</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 40px;
            background: #f5f5f5;
        }
        .container {
            background: white;
            padding: 25px;
            border-radius: 10px;
        }
        h1 {
            color: #333;
        }
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 10px;
        }
        th {
            background: #eeeeee;
```

Рисунок 3.8 – Автоматично згенерована html-сторінка результату для завдання «Traffic Light»

Крім результатів окремих перевірок, HTML-звіт (рис. 3.9) містить статистичну інформацію про кількість успішних перевірок, кількість виявлених помилок, підсумковий відсоток виконання завдання та загальний висновок щодо правильності виконаної роботи. Використання такого формату забезпечує більш наочне представлення результатів та спрощує аналіз отриманих даних.

Звіт перевірки Wokwi-проєкту

Перевірка	Статус	Повідомлення
Файл .ino	OK	Файл програмного коду знайдено
Файл diagram.json	OK	Файл схеми знайдено
wokwi-arduino-uno	OK	Компонент знайдено
wokwi-led	OK	Компонент знайдено
wokwi-resistor	OK	Компонент знайдено
LED	OK	Світлодіод знайдено
Підключення LED	OK	LED green підключено до піна 12 через резистор rg
Відповідність коду і схеми	OK	У коді знайдено використання піна 12 та функції digitalWrite
Код: setup	OK	Елемент setup знайдено
Код: loop	OK	Елемент loop знайдено
Код: pinMode	OK	Елемент pinMode знайдено
Код: digitalWrite	OK	Елемент digitalWrite знайдено
Код: delay	OK	Елемент delay знайдено

Успішних перевірок: 13

Помилок: 0

Загальний бал: 100.0%

Висновок: Завдання виконано правильно

Рисунок 3.9 – Результат перевірки 5-го завдання у вигляді html-сторінки

Результати проведеного тестування підтвердили працездатність розробленого прототипу системи автоматизованого тестування мікроконтролерних проєктів. У процесі дослідження система успішно виконала аналіз структури проєктів, електронних схем та програмного коду для всіх підготовлених тестових завдань.

Перевірка еталонних проєктів показала коректне визначення складу компонентів, правильності їх підключення та відповідності програмного коду вимогам завдання. Додатково було проведено тестування на проєктах із навмисно

внесеними помилками. У всіх досліджуваних випадках система успішно виявила невідповідності між схемою, програмним кодом та правилами тестування.

Отримані результати свідчать про можливість використання розробленого програмного засобу для автоматизації первинної перевірки навчальних робіт з програмування мікроконтролерних систем. Запропонований підхід дозволяє зменшити навантаження на викладача, підвищити об'єктивність оцінювання та скоротити час перевірки студентських проєктів.

Висновки до розділу 3

У третьому розділі було виконано розробку та дослідження прототипу системи автоматизованого тестування програмного забезпечення мікроконтролерних систем. На основі сформованих вимог реалізовано програмний засіб, призначений для аналізу навчальних проєктів, створених у середовищі моделювання Wokwi.

Розроблена система забезпечує завантаження проєктів у вигляді ZIP-архівів, аналіз структури електронної схеми та програмного коду, перевірку відповідності проєкту вимогам конкретного завдання, а також формування звітів про результати тестування. Для підвищення універсальності програмного забезпечення реалізовано механізм використання зовнішніх файлів правил, що дозволяє виконувати перевірку різних навчальних завдань без внесення змін до програмного коду системи.

У ході дослідження було підготовлено набір тестових проєктів, які моделюють типові завдання з програмування мікроконтролерних систем. Результати тестування показали, що розроблений прототип успішно виконує аналіз електронних схем, визначає наявність необхідних компонентів, перевіряє правильність їх підключення та контролює використання програмних конструкцій відповідно до вимог завдання.

Додатково було проведено перевірку проєктів із навмисно внесеними помилками, що дозволило оцінити здатність системи виявляти невідповідності між програмним кодом, електронною схемою та правилами тестування. Отримані

результати підтвердили ефективність запропонованого підходу та можливість використання системи для автоматизації первинної перевірки студентських робіт.

Отже, під час написання дипломної роботи, розроблений прототип, який дозволяє зменшити час перевірки навчальних проєктів, підвищити об’єктивність оцінювання та забезпечити єдині критерії контролю результатів навчання. Отримані результати можуть бути використані як основа для подальшого розвитку систем автоматизованого тестування мікроконтролерних проєктів та розширення функціональних можливостей програмного забезпечення.

4 ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

Охорона праці є невід’ємною складовою професійної діяльності фахівців у сфері інформаційних технологій, електроніки та автоматизації. Процес створення embedded-систем супроводжується низкою виробничих ризиків. До них належать електричні небезпеки під час роботи з лабораторними стендами та джерелами живлення, підвищене навантаження на органи зору, тривала робота за комп’ютером, а також психоемоційне навантаження, пов’язане з розробкою та налагодженням програмного забезпечення [27].

Основні законодавчі акти в галузі охорони праці

Правові та організаційні засади забезпечення безпеки праці в Україні визначаються законодавчими та нормативними документами, які регламентують права та обов’язки працівників і роботодавців, порядок організації безпечних умов праці, проведення інструктажів, навчання та оцінювання професійних ризиків [27, 28].

Основними нормативно-правовими актами у сфері охорони праці є:

- Кодекс законів про працю України;
- Закон України «Про охорону праці»;
- Закон України «Про загальнообов'язкове державне соціальне страхування»;
- ДСТУ ISO 45001:2019 «Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування»;
- Правила пожежної безпеки в Україні;
- державні санітарні норми та правила організації робочих місць користувачів персональних електронно-обчислювальних машин.

Відповідно до положень Кодексу законів про працю України роботодавець зобов’язаний створити безпечні та нешкідливі умови праці, забезпечити належний технічний стан обладнання та організувати контроль за дотриманням вимог охорони праці. Працівник, у свою чергу, повинен виконувати встановлені правила

безпеки, використовувати обладнання відповідно до інструкцій та негайно повідомляти про виявлені несправності або небезпечні ситуації [2829].

Положення Кодексу законів про працю України

Закон України «Про охорону праці» визначає основні принципи державної політики у сфері безпеки праці, серед яких пріоритет життя та здоров'я працівників, відповідальність роботодавця за створення належних умов праці та впровадження заходів щодо попередження виробничого травматизму. Закон також передбачає проведення інструктажів, навчання з питань охорони праці та систематичний контроль рівня професійних ризиків [29].

Для сучасних підприємств дедалі більшого значення набуває ризик-орієнтований підхід до організації охорони праці. Його основою є виявлення потенційних небезпек, оцінювання ймовірності їх виникнення та впровадження заходів щодо мінімізації можливих наслідків. Відповідні принципи реалізовані у міжнародному стандарті ДСТУ ISO 45001:2019, який визначає вимоги до систем управління охороною праці та безпекою персоналу (ISO 45001:2018 Occupational health and safety management systems — Requirements with guidance for use) [30].

Особливістю робіт, пов'язаних із розробкою та тестуванням мікроконтролерних систем, є поєднання програмної та апаратної складових. Розробник працює не лише з програмним кодом, а й з електронними модулями, макетними платами, джерелами живлення, засобами вимірювання та комп'ютерною технікою. У зв'язку з цим безпечна організація робочого місця повинна враховувати вимоги електробезпеки, пожежної безпеки та ергономіки.

Особлива увага приділяється організації робочого місця користувача персонального комп'ютера. Відповідно до сучасних рекомендацій з ергономіки, робоче місце повинно забезпечувати правильне розташування монітора, клавіатури та інших засобів введення інформації, достатній рівень освітлення та можливість зміни робочої пози протягом дня. Недотримання цих вимог може призводити до перевтоми, погіршення зору та розвитку професійних захворювань опорно-рухового апарату.

Крім традиційних виробничих ризиків, в умовах сучасної України необхідно враховувати надзвичайні ситуації техногенного та воєнного характеру. Організація безпечної роботи персоналу повинна передбачати порядок дій під час повітряних тривог, аварійних відключень електроенергії та інших надзвичайних ситуацій, які можуть впливати на безпеку працівників та збереження обладнання [32].

Таким чином, організаційно-правові основи охорони праці під час розробки та тестування програмного забезпечення мікроконтролерних систем базуються на вимогах чинного законодавства України, міжнародних стандартах управління безпекою праці та принципах ризик-орієнтованого підходу. Їх дотримання створює передумови для безпечної роботи персоналу, зниження професійних ризиків та підвищення ефективності виробничої діяльності.

4.2. Характеристика об’єкта та виявлення потенційних небезпек

Об'єктом дослідження в даній роботі є робоче місце розробника систем автоматизованого тестування програмного забезпечення мікроконтролерних систем. У процесі виконання професійних обов'язків фахівець здійснює розробку програмного забезпечення, моделювання електронних схем, тестування мікроконтролерних проєктів, аналіз результатів роботи програм та підключення лабораторного обладнання.

Основними технічними засобами, що використовуються під час виконання робіт, є персональний комп’ютер чи ноутбук, монітори, мережеве обладнання, джерела живлення, макетні плати, мікроконтролерні модулі та периферійні пристрої. Робота виконується переважно у приміщенні та характеризується тривалим використанням комп’ютерної техніки, високою концентрацією уваги та взаємодією з електронним обладнанням.

Під час розробки та тестування програмного забезпечення можуть виникати небезпечні та шкідливі виробничі фактори, що впливають на безпеку працівника та працездатність обладнання.

Таблиця 4.1 – Потенційні небезпеки під час експлуатації ПЗ

Небезпечний фактор	**Причина виникнення**	**Можливі наслідки**
Електричний струм	Пошкодження ізоляції, помилки підключення обладнання	Ураження електричним струмом
Тривала робота за ПК	Багатогодинна робота без перерв	Перевтома, захворювання опорно-рухового апарату
Навантаження на зір	Тривала робота з моніторами	Втома очей, погіршення зору
Психоемоційне навантаження	Висока концентрація уваги, пошук помилок	Стрес, зниження працездатності
Коротке замикання	Неправильне підключення електронних компонентів	Пошкодження обладнання, пожежа
Перегрів електронних модулів	Несправність компонентів або джерел живлення	Пожежна небезпека
Перебої електроживлення	Аварійні ситуації в мережі	Втрата даних, пошкодження обладнання
Надзвичайні ситуації	Воєнні дії, техногенні аварії	Загроза життю та здоров’ю працівників

4.2.1. Електричні небезпеки

Під час розробки та тестування мікроконтролерних систем використовуються персональні комп’ютери, блоки живлення, програматори та електронні модулі. Незважаючи на використання переважно низьковольтного обладнання, існує ризик ураження електричним струмом у разі пошкодження ізоляції, несправності мережевого обладнання або неправильного підключення електронних компонентів [34, **Error! Reference source not found.**].

Особливу увагу необхідно приділяти стану кабелів живлення, справності блоків живлення та дотриманню вимог електробезпеки під час роботи з лабораторними стендами.

4.2.2. Ергономічні небезпеки

Розробка програмного забезпечення передбачає тривале перебування працівника у сидячому положенні. Недостатньо ергономічне робоче місце може призводити до перенавантаження м’язів спини, шиї та верхніх кінцівок.

До основних факторів належать:

- неправильне положення монітора;
- невідповідність висоти робочого столу;
- відсутність підтримки попереку;
- обмежена рухова активність.

Тривалий вплив цих факторів може сприяти розвитку захворювань опорно-рухового апарату та зниженню працездатності працівника [36].

4.2.3. Навантаження на органи зору

У процесі програмування та аналізу результатів тестування працівник тривалий час взаємодіє з моніторами. Вплив штучного освітлення, неправильні налаштування яскравості дисплея та тривала концентрація на дрібних елементах інтерфейсу можуть викликати втому очей.

Для зниження навантаження необхідно забезпечувати достатній рівень освітлення робочої зони, дотримуватися режиму праці та відпочинку, а також використовувати монітори з відповідними параметрами відображення інформації [37].

4.2.4. Психофізіологічні фактори

Робота розробника програмного забезпечення пов’язана з постійною інтелектуальною діяльністю, аналізом великих обсягів інформації та пошуком помилок у програмному коді. Під час тестування мікроконтролерних систем додатково виникає необхідність аналізу взаємодії між програмним та апаратним забезпеченням.

До несприятливих факторів належать:

- висока концентрація уваги;

- значне розумове навантаження;
- робота в умовах обмежених термінів;
- необхідність багаторазового повторення тестових процедур.

Тривалий вплив таких факторів може спричиняти підвищену втому та зниження продуктивності праці [38].

4.2.5. Пожежна небезпека

Особливу увагу необхідно приділяти роботі з макетними платами та джерелами живлення. Помилки під час підключення електронних компонентів можуть призводити до перевищення допустимих струмів, перегрівання провідників, виходу з ладу електронних модулів та виникнення локальних осередків займання.

Відповідно до вимог Закону України «Про охорону праці» роботодавець зобов'язаний забезпечити належний технічний стан обладнання, контроль безпечної експлуатації електроустановок та створення умов, що унеможливлюють виникнення пожеж через виробничі фактори [29].

Згідно з вимогами Правил пожежної безпеки в Україні, усі приміщення, у яких експлуатується електронне обладнання, повинні бути забезпечені первинними засобами пожежогасіння, евакуаційними шляхами та засобами оповіщення про пожежу [32].

Для робочих місць розробників електронних систем особливо важливими є такі профілактичні заходи:

- періодична перевірка стану електропроводки;
- використання сертифікованих блоків живлення;
- контроль температурного режиму обладнання;
- недопущення перевантаження мережевих фільтрів та подовжувачів;
- вимкнення обладнання після завершення роботи;
- регулярне технічне обслуговування комп’ютерної техніки;
- використання джерел безперебійного живлення для критично важливих пристроїв.

Відповідно до міжнародного стандарту ISO 45001:2018 роботодавець повинен здійснювати постійне оцінювання виробничих ризиків та впроваджувати заходи щодо попередження аварійних ситуацій, у тому числі пов'язаних із пожежною небезпекою [39].

4.2.6. Воєнні та надзвичайні ризики

В умовах сучасної України необхідно враховувати ризики, пов’язані з воєнними діями та надзвичайними ситуаціями. До них належать повітряні тривоги, аварійні відключення електроенергії, пошкодження телекомунікаційної інфраструктури та інші фактори, що можуть створювати загрозу життю та здоров’ю працівників.

Організація роботи повинна передбачати наявність плану евакуації, порядок дій під час сигналів оповіщення та резервні засоби збереження даних у разі аварійних ситуацій.

4.3. Дослідження ризику реалізації потенційних небезпек на об’єкті проектування та розробка заходів щодо їх попередження

Ефективне функціонування системи охорони праці передбачає своєчасне виявлення потенційних небезпек, оцінювання ймовірності їх виникнення та визначення заходів щодо мінімізації можливих наслідків. Для цього застосовується ризик-орієнтований підхід, який рекомендований стандартом ДСТУ ISO 45001:2019 та широко використовується під час організації безпечних умов праці [30].

У межах даної роботи було проведено аналіз потенційних небезпек, характерних для робочого місця розробника систем автоматизованого тестування програмного забезпечення мікроконтролерних систем. Оцінювання здійснювалося на основі двох критеріїв:

- ймовірності виникнення небезпечної події;
- тяжкості можливих наслідків.

Результати оцінювання наведено в табл. 4.2.

Таблиця 4.2 – Результати оцінювання

Небезпечна ситуація	**Можливі наслідки**	**Ймовірність виникнення**	**Рівень ризику**
Ураження електричним струмом при роботі з обладнанням	Травмування працівника	Низька	Високий
Коротке замикання електронних компонентів	Пошкодження обладнання, пожежа	Середня	Середній
Тривала робота за комп'ютером	Порушення постави, захворювання опорно-рухового апарату	Висока	Середній
Підвищене навантаження на органи зору	Втома очей, зниження працездатності	Висока	Середній
Психоемоційне перевантаження	Стрес, перевтома	Середня	Середній
Перебої електроживлення	Втрата даних, зупинка роботи	Середня	Низький
Пожежа електрообладнання	Загроза життю працівників, пошкодження майна	Низька	Високий
Надзвичайні ситуації та воєнні ризики	Загроза життю та здоров’ю персоналу	Низька	Високий

Аналіз отриманих результатів показує, що найбільш критичними є ризики, пов'язані з електробезпекою, пожежною безпекою та надзвичайними ситуаціями. Незважаючи на відносно низьку ймовірність їх виникнення, потенційні наслідки можуть бути значними як для працівників, так і для матеріальних ресурсів підприємства.

Для зменшення ризику ураження електричним струмом необхідно використовувати лише справне обладнання, проводити періодичний контроль стану електромережі та дотримуватися вимог безпечної експлуатації електротехнічних пристроїв. Роботи з підключення або модифікації електронних схем повинні виконуватися при відключеному живленні.

Окрему увагу необхідно приділяти заходам реагування на надзвичайні ситуації. Працівники повинні бути ознайомлені з планами евакуації, порядком дій під час повітряної тривоги та місцями розташування найближчих укриттів. Для збереження результатів роботи доцільно використовувати резервне копіювання даних та хмарні сервіси зберігання інформації.

Таким чином, проведене оцінювання показало, що більшість виявлених ризиків можуть бути ефективно знижені шляхом впровадження організаційних та технічних заходів безпеки. Виконання зазначених рекомендацій сприятиме створенню безпечних умов праці під час розробки та тестування програмного забезпечення мікроконтролерних систем.

Висновки до розділу 4

У даному розділі було розглянуто організаційно-правові аспекти забезпечення охорони праці під час розробки та тестування програмного забезпечення мікроконтролерних систем. Проведений аналіз нормативно-правової бази показав, що безпечна організація праці повинна здійснюватися відповідно до вимог чинного законодавства України, міжнародних стандартів у сфері управління професійними ризиками та нормативних документів з пожежної й електричної безпеки.

Було охарактеризовано об’єкт проєктування та визначено основні небезпечні й шкідливі фактори, які можуть виникати під час роботи розробника embedded-систем. До найбільш суттєвих ризиків належать ураження електричним струмом, пожежна небезпека, тривала робота за комп’ютером, підвищене навантаження на органи зору, психоемоційне перевантаження та вплив надзвичайних ситуацій.

У результаті проведеної оцінки ризиків встановлено, що найбільшу потенційну небезпеку становлять пожежонебезпечні ситуації, аварії електрообладнання та надзвичайні події, які можуть призвести до значних матеріальних збитків і створювати загрозу життю та здоров’ю працівників. Для зниження рівня ризику запропоновано комплекс організаційних та технічних заходів, зокрема контроль технічного стану обладнання, дотримання вимог

електробезпеки, використання засобів пожежогасіння, забезпечення ергономічних умов праці та впровадження заходів щодо резервного збереження даних.

Таким чином, виконані дослідження підтверджують необхідність системного підходу до організації безпечних умов праці під час розробки та тестування програмного забезпечення мікроконтролерних систем. Реалізація запропонованих заходів дозволяє мінімізувати вплив небезпечних факторів, підвищити рівень безпеки працівників та забезпечити стабільність виконання професійної діяльності.

ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі було розглянуто проблему автоматизації тестування програмного забезпечення мікроконтролерних систем та досліджено сучасні підходи до розробки, налагодження і перевірки embedded-проєктів. Проведений аналіз показав, що зі збільшенням складності мікроконтролерних систем зростають вимоги до якості програмного забезпечення та ефективності процесів його тестування.

У першому розділі роботи було проаналізовано сучасні мікроконтролерні системи, сфери їх застосування та вимоги до надійності програмного забезпечення. Розглянуто етапи розробки та налагодження embedded-систем, а також виконано огляд існуючих засобів модульного тестування, моделювання та автоматизованого аналізу програмного коду. Проведене дослідження показало відсутність спеціалізованих засобів автоматизованої перевірки навчальних мікроконтролерних проєктів, що підтверджує актуальність розробки власного рішення.

У другому розділі сформульовано вимоги до системи автоматизованого тестування та розроблено її структуру. Визначено підходи до аналізу програмного коду та електронної схеми, обрано формати вхідних даних і структуру звітів за результатами перевірки. Запропонована архітектура системи дозволяє виконувати аналіз проєктів, створених у середовищі Wokwi, без необхідності використання фізичного обладнання.

У третьому розділі було розроблено програмний прототип системи автоматизованого тестування мовою Python. Реалізовано модулі завантаження та аналізу Wokwi-проєктів, обробки файлів програмного коду та опису електронної схеми, а також модуль формування результатів тестування. Проведене дослідження роботи системи підтвердило можливість автоматичного визначення відповідності проєкту встановленим вимогам та формування звітів за результатами перевірки.

У четвертому розділі розглянуто питання охорони праці під час розробки та тестування програмного забезпечення мікроконтролерних систем. Проаналізовано потенційні небезпечні фактори, виконано оцінювання професійних ризиків та запропоновано заходи щодо забезпечення безпечних умов праці.

Практичне значення отриманих результатів полягає у створенні прототипу системи автоматизованого тестування, який може використовуватися в навчальному процесі для перевірки лабораторних та практичних робіт з мікроконтролерних систем. Запропонований підхід дозволяє зменшити трудомісткість перевірки студентських проєктів, скоротити час оцінювання та підвищити об'єктивність результатів контролю.

Таким чином, поставлена мета дипломної роботи досягнута, а всі визначені завдання дослідження виконані в повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Novak, J., Kubik, D. (2021). Improving human-machine interaction in CNC control systems: Design approaches and case studies. International Journal of Manufacturing Systems, 48(4), 215–226.

2. AlMadhoun A. S. A. Microcontroller. Maker innovations series. Berkeley, CA, 2023. P. 1–55. URL: https://doi.org/10.1007/978-1-4842-9582-3_1 (date of access: 01.06.2026).

3. Pougnet P., El Hami A. Reliability-based design optimization. Embedded mechatronic systems. 2019. P. 1–27. URL: https://doi.org/10.1016/b978-1-78548-189-5.50001-x (date of access: 01.06.2026).

4. Bolanakis D. E. Micro-Controller fundamentals. Microcontroller education. Cham, 2018. P. 11–61. URL: https://doi.org/10.1007/978-3-031-79589-3_2 (date of access: 01.06.2026).

5. Microcontrollers. Auto electronics projects. 1995. P. 71–108. URL: https://doi.org/10.1016/b978-0-08-049963-5.50007-0 (date of access: 01.06.2026).

6. Hüning F. Mikrocontroller. Embedded systems für iot. Berlin, Heidelberg, 2018. P. 19–51. URL: https://doi.org/10.1007/978-3-662-57901-5_3 (date of access: 01.06.2026).

7. Developing a functional electrostimulation system using an arduino microcontroller. Proceedings of National Polytechnic University of Armenia. INFORMATION TECHNOLOGIES, ELECTRONICS, RADIO ENGINEERING. 2024. URL: https://doi.org/10.53297/18293336-2024.2-119 (date of access: 01.06.2026).

8. IoT Editorial Office I. Acknowledgment to the reviewers of iot in 2022. IoT. 2023. Vol. 4, no. 1. P. 56. URL: https://doi.org/10.3390/iot4010003 (date of access: 01.06.2026).

9. Pankov D. A., Denisova L. A. Automated testing and fault diagnosis of the microcontroller system. IOP conference series: materials science and engineering. 2019. Vol. 537. P. 022072. URL: https://doi.org/10.1088/1757-899x/537/2/022072 (date of access: 01.06.2026).

10. Microcontroller-based vein testing system / Z. A. Godo et al. 2017 9th international conference on electronics, computers and artificial intelligence (ECAI), Targoviste, 29 June – 1 July 2017. 2017. URL: https://doi.org/10.1109/ecai.2017.8166417 (date of access: 01.06.2026).

11. Sadraey M. H. Microcontroller. Unmanned aircraft design. Cham, 2017. P. 123–137. URL: https://doi.org/10.1007/978-3-031-79582-4_7 (date of access: 17.06.2026).

12. Corporation I., Intel. Oem boards and systems handbook. Intel Corporation (CA), 1989.

13. Pougnet P., El Hami A. Reliability-based design optimization. Embedded mechatronic systems. 2019. P. 1–27. URL: https://doi.org/10.1016/b978-1-78548-189-5.50001-x (date of access: 17.06.2026).

14. Nehete M., Rane A. Embedded computing - embedded systems book: embedded computing - embedded systems. Independently Published, 2021.

15. Debugging tools and firmware profiling / M. Rossi et al. Introduction to microcontroller programming for power electronics control applications. Boca Raton, 2021. P. 378–390. URL: https://doi.org/10.1201/9781003196938-20 (date of access: 17.06.2026).

16. Promotion boards. Staff reporting and staff development. 2017. P. 24–27. URL: https://doi.org/10.4324/9781315232652-4 (date of access: 17.06.2026).

17. Sultana M. T. Development of microcontroller. LAP Lambert Academic Publishing, 2012. 96 p.

18. Bergersen E. Embedded unit testing framework : thesis. 2013. URL: http://urn.kb.se/resolve?urn=urn:nbn:no:ntnu:diva-22986 (date of access: 18.06.2026).

19. Tao W. Y.-Y. A firmware data compression unit. Urbana : Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1974. 55 p.

20. Молчанов О. А., Сергієнко А. М., Романкевич В. О. Мікроконтролери зі стековою архітектурою. Problems of informatization and management. 2023. Т. 2, № 74. С. 74–80. URL: https://doi.org/10.18372/2073-4751.74.17885 (дата звернення: 18.06.2026).

21. Кучерук В., Кулаков П., Ліман В. Онлайн-сервіс wokwi як симулятор мікропроцесорних систем. Measuring and computing devices in technological processes. 2023. № 2. С. 153–158. URL: https://doi.org/10.31891/2219-9365-2023-74-19 (дата звернення: 18.06.2026).

22. Bryant S. C. Tinkercad for dummies. Wiley & Sons, Incorporated, John, 2018. 264 p.

23. Puji M. N., Astuti W. Simulation of data acquisition system using simulide, virtual serial port and python. 2023 9th international conference on education and technology (ICET), Malang, Indonesia, 7 October 2023. 2023. URL: https://doi.org/10.1109/icet59790.2023.10435244 (date of access: 18.06.2026).

24. Logic circuit validation. A fascinating country in the world of computing. 1999. P. 147–164. URL: https://doi.org/10.1142/9789812817952_0005 (date of access: 18.06.2026).

25. Благий А. Адаптивний статичний аналіз якості програмного коду. Modern engineering and innovative technologies. 2023. № 42-02. С. 28–36. URL: https://doi.org/10.30890/2567-5273.2025-42-02-035 (дата звернення: 18.06.2026).

26. Stapp L., Roman A., Pilaeten M. Foundation level syllabus. ISTQB® certified tester foundation level. Cham, 2023. P. 7–15. URL: https://doi.org/10.1007/978-3-031-42767-1_2 (date of access: 18.06.2026).

27. Кобилянський О. В., Дембіцька С. В., Кобилянська І. М. Охорона праці в галузі інформаційних технологій : навчальний посібник. Вінниця : ВНТУ, 2020. 208 с.

28. Кодекс законів про працю України : Закон України від 10.12.1971 № 322-VIII. URL: https://zakon.rada.gov.ua/laws/show/322-08 (дата звернення: 18.06.2026).

29. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 18.06.2026).

30. ДСТУ ISO 45001:2019. Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування. Київ : ДП «УкрНДНЦ», 2019.

31. ISO 45001:2018. Occupational health and safety management systems — Requirements with guidance for use. Geneva : International Organization for Standardization, 2018.

32. Правила пожежної безпеки в Україні : Наказ МВС України № 1417 від 30.12.2014. URL: https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 18.06.2026).

33. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. ДСанПіН 3.3.2.007-98.

34. Electromagnetic safety of workplaces equipped with computer hardware. Infokommunikacionnye tehnologii. 2020. P. 347–353. URL: https://doi.org/10.18469/ikt.2020.18.3.14 (date of access: 18.06.2026).

35. Rock B. D. A. Laboratory low voltage supply. Education + training. 1959. Vol. 1, no. 7. P. 27–32. URL: https://doi.org/10.1108/eb001571 (date of access: 18.06.2026).

36. Марич В. М., Мала Х. І. Ергономічні проблеми при роботі за комп'ютером : thesis. 2016. URL: http://hdl.handle.net/123456789/3175 (дата звернення: 18.06.2026).

37. Syndrome C. V. Computer vision syndrome. Nursing journal of india. 2009. Vol. C, no. 10. P. 236–237. URL: https://doi.org/10.48029/nji.2009.c1003 (date of access: 18.06.2026).

38. Occupational and visual stress among software engineers in north india: a cross-sectional study. Journal of ophthalmic pathology. 2026. Vol. 8, no. 1. P. 1–4. URL: https://doi.org/10.36266/jop/135 (date of access: 18.06.2026).

39. ISO 45001:2018. Occupational health and safety management systems - Requirements with guidance for use. Geneva : International Organization for Standardization, 2018.

40. Friend M. A., Kohn J. P. Fundamentals of Occupational Safety and Health. 7th ed. Lanham : Rowman & Littlefield, 2018.

41. Hughes P., Ferrett E. Introduction to Health and Safety at Work. 6th ed. Routledge, 2019.

ДОДАТОК А. Початковий код розробленої програми

Файл main.py – запускає основні процеси програми

```
from checker import check_wokwi_project
from report import save_report_txt
from report import save_report_txt, save_report_html

zip_path = "wokwi_checker/test/test_project.zip"

results = check_wokwi_project(zip_path)

print("ЗВІТ ПЕРЕВІРКИ")
print("-" * 40)

zip_path = r"D:\...\01_blink_led_d12.zip"
rules_path = r"D:\...\rules\01_blink_led_d12.json"

results = check_wokwi_project(zip_path, rules_path)

for name, status, message in results:
    print(f"{name}: {status} — {message}")

save_report_txt(results, "wokwi_checker/report_result.txt")
save_report_html(results, "wokwi_checker/report_result.html")
print("HTML-звіт збережено у файл: wokwi_checker/report_result.html")
print("\nЗвіт збережено у файл: wokwi_checker/report_result.txt")
```

Файл checker.py – робить основні перевірки

```
import zipfile
import json
import os
import tempfile

def check_code_uses_pin(code, pin, function_name="digitalWrite"):
    pin = str(pin)

    if pin in code and function_name in code:
        return (
            "Відповідність коду і схеми",
            "OK",
            f"У коді знайдено використання піна {pin} та функції {function_name}"
        )
```

```
    return (
        "Відповідність коду і схеми",
        "ERROR",
        f"У коді не знайдено використання піна {pin} або функції {function_name}"
    )

def find_part_ids_by_type(parts, component_type):
    return [
        part.get("id")
        for part in parts
        if part.get("type") == component_type
    ]

def are_connected(connections, point1, point2):
    for c in connections:
        if point1 in c[:2] and point2 in c[:2]:
            return True
    return False

def check_led_pin_universal(parts, connections, expected_pin="12"):
    led_ids = find_part_ids_by_type(parts, "wokwi-led")
    resistor_ids = find_part_ids_by_type(parts, "wokwi-resistor")

    for led_id in led_ids:
        for resistor_id in resistor_ids:
            variant_1 = (
                are_connected(connections, f"{led_id}:A", f"{resistor_id}:1")
                and are_connected(connections, f"{resistor_id}:2",
f"uno:{expected_pin}")
            )

            variant_2 = (
                are_connected(connections, f"{led_id}:A", f"{resistor_id}:2")
                and are_connected(connections, f"{resistor_id}:1",
f"uno:{expected_pin}")
            )

            led_to_pin_through_resistor = variant_1 or variant_2

            led_to_gnd = any(
                f"{led_id}:C" in c[:2] and "GND" in c[0] + c[1]
                for c in connections
            )

            if led_to_pin_through_resistor and led_to_gnd:
                return (
```

```
                    "Підключення LED",
                    "OK",
                    f"LED {led_id} підключено до піна {expected_pin} через резистор
{resistor_id}"
                )

    return (
        "Підключення LED",
        "ERROR",
        f"Не знайдено правильне підключення LED до піна {expected_pin} через резистор"
    )

def load_rules(rules_path):
    with open(rules_path, "r", encoding="utf-8") as f:
        return json.load(f)

# Функція перевірки підключення світлодіода до резистора та піну Arduino
def check_led_pin(connections, expected_pin="12"):
    led_connected_to_resistor = False
    resistor_connected_to_pin = False

    for c in connections:
        point_a = c[0]
        point_b = c[1]

        if ("led1:A" in [point_a, point_b]) and ("r1:1" in [point_a, point_b]):
            led_connected_to_resistor = True

        if ("r1:2" in [point_a, point_b]) and (f"uno:{expected_pin}" in [point_a,
point_b]):
            resistor_connected_to_pin = True

    if led_connected_to_resistor and resistor_connected_to_pin:
        return ("Підключення LED", "OK", f"LED підключено до піна {expected_pin} через
резистор")
    else:
        return ("Підключення LED", "ERROR", f"LED не підключено до очікуваного піна
{expected_pin}")

   # Функція перевірки Wokwi-проєкту
def check_wokwi_project(zip_path, rules_path):
    results = []
    rules = load_rules(rules_path)

    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_path, 'r') as archive:
```

```
            archive.extractall(temp_dir)

        ino_file = None
        diagram_file = None

        for root, dirs, files in os.walk(temp_dir):
            for file in files:
                if file.endswith(".ino"):
                    ino_file = os.path.join(root, file)
                if file == "diagram.json":
                    diagram_file = os.path.join(root, file)

        if ino_file:
            results.append(("Файл .ino", "OK", "Файл програмного коду знайдено"))
        else:
            results.append(("Файл .ino", "ERROR", "Файл програмного коду не
знайдено"))

        if diagram_file:
            results.append(("Файл diagram.json", "OK", "Файл схеми знайдено"))
        else:
            results.append(("Файл diagram.json", "ERROR", "Файл схеми не знайдено"))

        if not ino_file or not diagram_file:
            return results

        with open(diagram_file, "r", encoding="utf-8") as f:
            diagram = json.load(f)

        with open(ino_file, "r", encoding="utf-8") as f:
            code = f.read()

        parts = diagram.get("parts", [])
        connections = diagram.get("connections", [])

        print("\n3'єднання:")
        for c in connections:
         print(c)

        part_types = [part.get("type", "") for part in parts]

        print("\nЗнайдені компоненти:")
        for p in part_types:
            print(p)

        for component in rules["required_components"]:
```

```
            if component in part_types:
                results.append(
                (component, "OK", "Компонент знайдено")
                )
            else:
                results.append(
                (component, "ERROR", "Компонент не знайдено")
            )

        if any("led" in part_type for part_type in part_types):
            results.append(("LED", "OK", "Світлодіод знайдено"))
        else:
            results.append(("LED", "ERROR", "Світлодіод не знайдено"))

        results.append(
            check_led_pin_universal(
                parts,
                connections,
                expected_pin=rules.get("expected_led_pin", "12")
            )
        )

        results.append(
            check_code_uses_pin(
                code,
                rules.get("expected_led_pin", "12"),
                "digitalWrite"
            )
        )

        #  if any("pushbutton" in part_type or "button" in part_type for part_type in
part_types):
        #     results.append(("Кнопка", "OK", "Кнопку знайдено"))
        # else:
        #     results.append(("Кнопка", "ERROR", "Кнопку не знайдено"))

        required_code_elements = rules.get("required_code_elements", [])

        for element in required_code_elements:
            if element in code:
                results.append((f"Код: {element}", "OK", f"Елемент {element}
знайдено"))
            else:
                results.append((f"Код: {element}", "ERROR", f"Елемент {element} не
знайдено"))
        return results
```

Файл gui.py – файл графічного інтерфейсу, генерує інтерфейс, де користувач може взаємодіяти з програмою

```
import sys
from PyQt6.QtWidgets import (
    QApplication, QWidget, QPushButton, QVBoxLayout,
    QTextEdit, QFileDialog, QLabel
)

from checker import check_wokwi_project
from report import save_report_txt, save_report_html

class WokwiCheckerApp(QWidget):
    def __init__(self):
        super().__init__()

        self.rules_path = None
        self.zip_path = None
        self.results = []

        self.setWindowTitle("Wokwi Project Checker")
        self.resize(800, 600)

        self.label = QLabel("Оберіть ZIP-проєкт та JSON-файл правил")

        self.select_button = QPushButton("Вибрати ZIP")
        self.select_rules_button = QPushButton("Вибрати файл правил")
        self.check_button = QPushButton("Запустити перевірку")
        self.save_txt_button = QPushButton("Зберегти TXT-звіт")
        self.save_html_button = QPushButton("Зберегти HTML-звіт")

        self.output = QTextEdit()
        self.output.setReadOnly(True)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.select_button)
        layout.addWidget(self.select_rules_button)
        layout.addWidget(self.check_button)
        layout.addWidget(self.save_txt_button)
        layout.addWidget(self.save_html_button)
        layout.addWidget(self.output)

        self.setLayout(layout)

        self.select_button.clicked.connect(self.select_zip)
```

```
        self.select_rules_button.clicked.connect(self.select_rules)
        self.check_button.clicked.connect(self.run_check)
        self.save_txt_button.clicked.connect(self.save_txt_report)
        self.save_html_button.clicked.connect(self.save_html_report)

    def select_zip(self):
        file_path, _ = QFileDialog.getOpenFileName(
            self,
            "Оберіть ZIP-файл",
            "",
            "ZIP files (*.zip)"
        )

        if file_path:
            self.zip_path = file_path
            self.label.setText(f"Обрано ZIP: {file_path}")

    def select_rules(self):
        file_path, _ = QFileDialog.getOpenFileName(
            self,
            "Оберіть JSON-файл правил",
            "",
            "JSON files (*.json)"
        )

        if file_path:
            self.rules_path = file_path
            self.label.setText(f"Обрано правила: {file_path}")

    def run_check(self):
        if not self.zip_path:
            self.output.setText("Спочатку оберіть ZIP-файл проєкту.")
            return

        if not self.rules_path:
            self.output.setText("Спочатку оберіть JSON-файл правил.")
            return

        self.results = check_wokwi_project(self.zip_path, self.rules_path)

        text = "ЗВІТ ПЕРЕВІРКИ\n"
        text += "-" * 40 + "\n"

        for name, status, message in self.results:
            text += f"{name}: {status} — {message}\n"
```

```
        self.output.setText(text)

    def save_txt_report(self):
        if not self.results:
            self.output.setText("Спочатку виконайте перевірку.")
            return

        file_path, _ = QFileDialog.getSaveFileName(
            self,
            "Зберегти TXT-звіт",
            "report_result.txt",
            "Text files (*.txt)"
        )

        if file_path:
            save_report_txt(self.results, file_path)
            self.output.append(f"\nTXT-звіт збережено: {file_path}")

    def save_html_report(self):
        if not self.results:
            self.output.setText("Спочатку виконайте перевірку.")
            return

        file_path, _ = QFileDialog.getSaveFileName(
            self,
            "Зберегти HTML-звіт",
            "report_result.html",
            "HTML files (*.html)"
        )

        if file_path:
            save_report_html(self.results, file_path)
            self.output.append(f"\nHTML-звіт збережено: {file_path}")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = WokwiCheckerApp()
    window.show()
    sys.exit(app.exec())
```

Файл report.py – файл, що генерує репорт у вигляді html-сторінки та текстовий репорт.

```
def save_report_html(results, output_path="report.html"):
    ok_count = sum(1 for _, status, _ in results if status == "OK")
    error_count = sum(1 for _, status, _ in results if status == "ERROR")
    total = ok_count + error_count

    score = 0
    if total > 0:
        score = round((ok_count / total) * 100, 2)

    conclusion = "Завдання виконано правильно" if error_count == 0 else "Проєкт
потребує виправлення"

    html = f"""
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Звіт перевірки Wokwi-проєкту</title>
    <style>
        body {{
            font-family: Arial, sans-serif;
            margin: 40px;
            background: #f5f5f5;
        }}
        .container {{
            background: white;
            padding: 25px;
            border-radius: 10px;
        }}
        h1 {{
            color: #333;
        }}
        table {{
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }}
        th, td {{
            border: 1px solid #ddd;
            padding: 10px;
        }}
        th {{
            background: #eeeeee;
```

```
        }}
        .ok {{
            color: green;
            font-weight: bold;
        }}
        .error {{
            color: red;
            font-weight: bold;
        }}
        .summary {{
            margin-top: 25px;
            padding: 15px;
            background: #f0f0f0;
            border-radius: 8px;
        }}
    </style>
</head>
<body>
<div class="container">
    <h1>Звіт перевірки Wokwi-проєкту</h1>

    <table>
        <tr>
            <th>Перевірка</th>
            <th>Статус</th>
            <th>Повідомлення</th>
        </tr>
"""

    for name, status, message in results:
        css_class = "ok" if status == "OK" else "error"

        html += f"""
        <tr>
            <td>{name}</td>
            <td class="{css_class}">{status}</td>
            <td>{message}</td>
        </tr>
"""

    html += f"""
    </table>

    <div class="summary">
        <p><strong>Успішних перевірок:</strong> {ok_count}</p>
        <p><strong>Помилок:</strong> {error_count}</p>
```

```
        <p><strong>Загальний бал:</strong> {score}%</p>
        <p><strong>Висновок:</strong> {conclusion}</p>
    </div>
</div>
</body>
</html>
"""

    with open(output_path, "w", encoding="utf-8") as f:
        f.write(html)

# Функція збереження звіту у текстовий файл
def save_report_txt(results, output_path="report.txt"):
    with open(output_path, "w", encoding="utf-8") as f:
        f.write("ЗВІТ ПЕРЕВІРКИ WOKWI-ПРОЄКТУ\n")
        f.write("=" * 40 + "\n\n")

        ok_count = 0
        error_count = 0

        for name, status, message in results:
            f.write(f"{name}: {status} — {message}\n")

            if status == "OK":
                ok_count += 1
            elif status == "ERROR":
                error_count += 1

        f.write("\n")
        f.write("ПІДСУМОК\n")
        f.write("-" * 40 + "\n")
        f.write(f"Успішних перевірок: {ok_count}\n")
        f.write(f"Помилок: {error_count}\n")
        score = ok_count / (ok_count + error_count) * 100
        f.write(f"Загальний бал: {score:.2f}%\n")
        if error_count == 0:
            f.write("Загальний висновок: завдання виконано правильно.\n")
        else:
            f.write("Загальний висновок: проєкт потребує виправлення.\n")
```