Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“ Програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів”

Виконав здобувач вищої освіти
IV курсу, групи КІ-20
ОПП «Комп'ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Талмазан С. Д.
«____» _____ 20 р.

Керівник проекту
доцент
_____ Коваленко А. С.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Талмазану Сергію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів*

керівник роботи *Коваленко Анна Степанівна, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№131-02 від 01.04.2024 року*

2. Строк подання студентом роботи до захисту *01.04.2024 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів*

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

1. Призначення та область використання.
2. Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 17 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	19.05.2024 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Талмазан С.Д. Програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення системи апаратного комплексу отримання, обробки та зберігання біострумів.

Метою роботи є розробка програмного забезпечення для системи апаратного комплексу отримання, обробки та зберігання біострумів на основі платформи ESP32 DEVKIT V1.

Результат роботи – програмна реалізація системи апаратного комплексу отримання, обробки та зберігання біострумі.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на платформі Windows 10/11.

Програму розроблено на мові програмування Arduino.

Ключові слова: інтернет речей, моніторинг, електрокардіографія, ESP32, AD8232, програмне забезпечення.

ABSTRACT

Talmazan S.D. Software of the hardware complex for receiving, processing and storing biocurrents. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this bachelor's thesis, the software of the system of hardware complex for receiving, processing and storing biocurrents was developed.

The aim of the work is to develop software for the system of hardware complex for receiving, processing and storing biocurrents based on the ESP32 DEVKIT V1 platform.

The result of the work is a software implementation of the system of hardware complex for receiving, processing and storing biocurrents.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was carried out. The proprietary software was developed and implemented, and all its components were described.

A user-friendly interface was developed. Instructions for working with the software are provided.

The software can be used on the Windows 10/11 platform.

The software is developed in the Arduino programming language.

Keywords: Internet of Things, monitoring, electrocardiography, ESP32, AD8232, software.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ.....2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 4
1.1 Призначення системи..... 5
1.2 Область застосування..... 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 5
2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 9
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 22
2.3 Розгорнута постановка завдання 32
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 9
3.1 Опис функціонування системи 34
3.2 Розробка структурної схеми..... 37
3.3 Розробка функціональної схеми 39
3.4 Розробка діаграми процесів..... 40
4 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ..... 34
4.1 Розробка блок-схем та опис алгоритмів функціонування системи 42
4.2 Захист розробленого програмного забезпечення..... 51
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ 53
6 ОСНОВНІ ВИСНОВКИ..... 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61

					ВКРБ-123.24.0015.00.00.ПЗ			
Вим	Арк.	№ докум.	Підп.	Дата				
Розроб.		Талмазан С.Д.			Програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів	Літ.	Аркуш	Аркушів
Перев.		Коваленко А.С.				Б	1	66
						КІ-20		
Н.контр.		Коваленко А.С.						
Затв.		Смірнов О.А.						

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БР – Бакалаврська робота

ПЗ – Програмне забезпечення

ПК – Персональний комп’ютер

IoT – Інтернет речей (Internet of Things)

IDE – Інтегроване середовище розробки

MQTT – Message Queue Telemetry Transport

ЕКГ – Електрокардіографія

ЕКС – Електрокардіостимулятор

АЦП – Аналого-цифровий перетворювач

і т.д. – і так далі

ВСТУП

Актуальність теми. Сучасне суспільство стрімко розвивається, і дослідження в галузі біострумів набувають все більшого значення. Біоструми – це електричні сигнали, що генеруються живими організмами і містять цінну інформацію про стан і функції організму.

Актуальність цієї теми зумовлена необхідністю забезпечення точності, ефективності та безпеки при зборі та аналізі біологічних сигналів. Враховуючи постійний розвиток медичних технологій та методів діагностики, біоімпедансне програмне забезпечення стало надзвичайно важливим. Ефективне використання цього програмного забезпечення значно підвищує якість медичної діагностики та сприяє швидкій і точній інтерпретації отриманих даних.

Україна, яка активно розвиває свою наукову та медичну галузь, має великий інтерес до вдосконалення методів обробки біологічних сигналів. Розробка відповідного програмного забезпечення може значно полегшити та прискорити діагностичний процес у медичних закладах України та сприяти розвитку медицини і підвищенню якості охорони здоров'я в країні.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи ЕКГ моніторингу людини, яка забезпечуватиме точний і надійний збір та аналіз електричних сигналів серця. Головною метою є створення функціонального пристрою, який забезпечуватиме моніторинг серцевої активності у реальному часі.

Щоб досягти цілі, було визначено план дослідження, який включає в себе ряд завдань:

– Дослідження основних принципів роботи мікроконтролера ESP32 DEVKITV1 та датчика ЕКГ AD8232.

– Створення ПЗ для зчитування інформації з датчика AD8232 та її передачі на спеціальний веб-сайт для моніторингу.

– Перевірка функціональності системи та здійснення її налагодження.

Практична цінність отриманих результатів. Результати цієї роботи мають велике практичне значення. Розроблене ПЗ може бути використане в різних галузях, включаючи медицину, біологію табіотехнології. В медицині, це може допомогти в діагностиці та лікуванні різних хвороб,оскільки біоструми відіграють важливу роль в багатьох біологічних процесах. В біології та біотехнологіях, це може поліпшити якість досліджень, пов’язаних з вивченням біострумів.

Це ПЗ готове до використання і може бути масштабоване для використання в різних установах та організаціях. В Україні, де біотехнологічна індустрія продовжує розвиватися, це може мати значний вплив на розвиток цієї галузі. Загалом, результати дослідження мають великий потенціал для практичного застосування в різних сферах. Подальші кроки у впровадженні цих результатів можуть сприяти покращенню якості життя та розвитку суспільства.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Дана система має за мету не лише моніторинг біострумів в тілі людини, а й надання надійного та зручного засобу для вимірювання та аналізу їхньої активності. Основним призначенням цієї системи є забезпечення процесів отримання, обробки та зберігання біострумів з метою моніторингу функціонального стану серця та інших важливих параметрів організму.

Унікальність цієї системи полягає в тому, що всі отримані показники з датчика передаються безпосередньо по Wi-Fi на спеціальний веб-сайт Ubidots. Це хмарна платформа для збору, візуалізації та аналізу даних з датчиків і IoT-пристроїв. Вона дозволяє користувачам моніторити в реальному часі, створювати графіки, налаштовувати сповіщення та реагувати на події на основі отриманих даних. Ubidots широко використовується в індустріальних та комерційних застосуваннях для управління системами моніторингу та керування. На цьому сайті дані піддаються обробці та аналізу, а результати відображаються у вигляді лінійної діаграми. Такий підхід забезпечує швидкий доступ до даних та дозволяє зручно відслідковувати динаміку змін в показниках біострумів.

Такий інтерфейс взаємодії з системою дозволяє не лише отримувати дані про електричну активність серця людини, але й аналізувати їх у реальному часі, виявляти аномалії та реагувати на них вчасно. В результаті будується ЕКГ серця, що є важливим інструментом для діагностики та моніторингу серцево-судинних захворювань, а також для підтримки загального стану здоров'я.

Основний функціонал системи полягає у наступному:

– Вимірювання електричної активності серця.

– Фільтрація та забезпечення високої якості сигналу завдяки датчику

AD8232.

– Отримання та обробка даних.

– Передача даних на спеціальний веб-сайт та візуалізація їх у вигляді ЕКГ.

1.2 Область застосування

Відазу треба зазначити, що розроблена система, яка використовує датчик AD8232, слід розглядати як прототип для дослідницьких цілей, а не як основу для професійного медичного обладнання. Незважаючи на це, цей прототип має великий потенціал для подальшого розвитку і використання у різних галузях.

Подальший розвиток системи може включати в себе удосконалення алгоритмів обробки даних, підвищення точності та надійності вимірювань, а також розширення функціональності для більш широкого спектру застосувань.

Наприклад, з удосконаленням алгоритмів обробки даних та збільшенням точності вимірювань, система може бути використана для медичної діагностики з більшою достовірністю результатів. Також, потрібно замінювати сам датчик на датчики які використовують у професійному обладнані.

Подальший розвиток поточного прототипа може призвести до створення потужної та багатофункціональної системи, яка знайде широке застосування у різних сферах життя та діяльності таких як:

– Медична діагностика та лікування: однією з головних областей застосування є медицина. ПЗ може використовуватися для виявлення та аналізу біологічних сигналів, що дозволяє лікарям проводити точну діагностику різноманітних захворювань серця, нервової системи та інших органів, а також для моніторингу ефективності лікування.

– Наукові дослідження: ПЗ може бути використане у наукових

дослідженнях у галузі біофізики, біології, медицини та інших суміжних областях. Воно дозволить дослідникам збирати та аналізувати біологічні дані для виявлення нових закономірностей та розробки нових методів діагностики та лікування.

– Фармацевтична промисловість: у фармацевтичній галузі ПЗ може бути використане для тестування ефективності лікарських засобів, контролю якості та безпеки медичних препаратів.

– Біометрика та безпека: використання біологічних сигналів для ідентифікації особи (біометричні дані) стає все більш поширеним у системах безпеки, доступу та аутентифікації.

– Спортивна медицина та фітнес: в ПЗ можуть бути реалізовані функції для моніторингу фізичного стану та навантаження під час тренувань, а також для виявлення потенційних проблем зі здоров'ям під час занять спортом.

– Вивчення психофізіологічних станів: біоструми можуть бути використані для вивчення психофізіологічних станів людини, таких як стрес, стан або настрій. ПЗ може аналізувати ці дані для діагностики та моніторингу психічного здоров'я.

– Оцінка фізичного стану у спеціальних умовах: у випадку роботи в екстремальних умовах, таких як космос або під водою, ПЗ може використовуватися для моніторингу фізіологічних параметрів та стану організму.

– Системи допомоги при фізичному та психологічному втомленні: ПЗ може бути використане для розробки систем, що допомагають виявляти та запобігати фізичному та психологічному втомленню у водіїв автомобілів, пілотів, операторів та інших професійних категорій.

Ці області застосування відображають лише деякі з потенційних сфер використання. Цей широкий спектр можливостей підкреслює значення і універсальність такої системи в різних аспектах життя та професійної діяльності. Від медичних досліджень до практичного застосування у спорті,

виробництві та безпеці, програмне забезпечення апаратного комплексу для біострумів стає важливим інструментом для забезпечення точності, ефективності та зручності у зборі та аналізі біологічних даних. Ці технології відкривають нові можливості для діагностики, моніторингу та підтримки здоров'я, сприяючи прогресу в науці, медицині та інших галузях, де вони можуть бути використані.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

На ринку доступний широкий вибір рішень щодо рішень ЕКГ обладнання, пристроїв та систем, які розробляють компанії, спеціалізовані в цій галузі. Це означає, що клієнти можуть вибирати серед різноманітних моделей з різними функціональними можливостями та характеристиками, що відповідають їхнім потребам та вимогам.

Серед переваг такого обладнання можна виділити його широкий функціонал, який може включати аналіз ритму серця, визначення патологічних змін у кардіограмі, інтеграцію з іншими медичними системами тощо. Однак, до недоліків відносять складність розшифровки кардіограм та високу вартість обладнання, яка може бути значним чинником обмеження для бюджетів медичних установ. Незважаючи на це, наявність різних моделей дозволяє вибрати обладнання, яке відповідає потребам конкретної клініки чи лікаря, забезпечуючи оптимальне співвідношення ціни та якості.

Однією з таких систем моніторингу є DiaCard[14], українська холтерівська платформа для кардіологічного тестування. Система безперервно записує ЕКГ від 2 до 12 відведень протягом 168 годин, які потім автоматично аналізуються системою обробки.

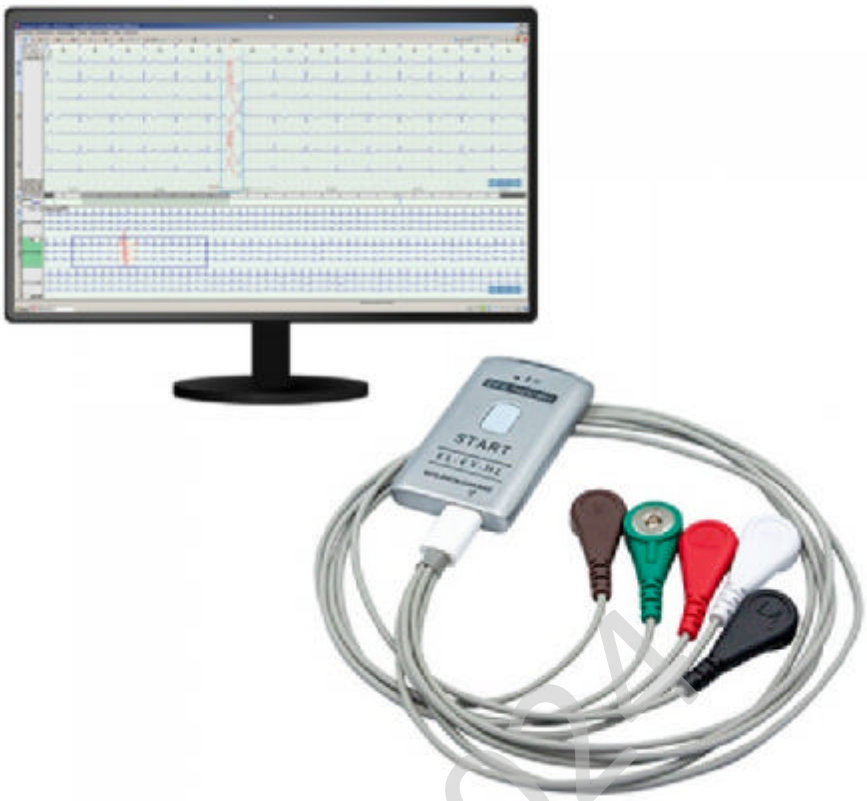

Рисунок 2.1 – Система моніторингу «DIACARD» 2.1 (LE)[51]

Diacard пропонує розширені можливості для досліджень, такі як аналіз артеріального тиску в бімодальному режимі, вимірювання дихання через імпедансну біполярну реографію, визначення рівня кисню у крові та оцінку роботи імплантованих кардіостимуляторів. Крім того, вона може моніторити активність пацієнта в реальному часі.

Це система, що складається з двох компонентів: портативний пристрій (один або декілька реєстраторів), який записує ЕКС, та система обробки, наприклад, ПК зі спеціалізованим програмним забезпеченням.

Для запису ЕКС та інших діагностичних параметрів використовуються портативні реєстратори (реєстратори), що дозволяють фіксувати дані в режимі реального часу і зберігати їх у внутрішній пам'яті. Після закінчення обстеження дані з реєстратора передаються на ПК, де вони аналізуються і робляться висновки.

Отримані дані та результати обробки зберігаються в архіві і можуть бути використані як для локального, так і для віддаленого архівування. Всі без винятку реєстратори можуть бути запрограмовані та активовані для тестування з ПК або автономно (з параметрами за замовчуванням).

Всі моделі реєстраторів мають такі функції, як візуальний контроль вхідних даних, обриву електродів і наявності підключених датчиків, розряду і відновлення батареї, що робить роботу з реєстратором максимально надійною і в той же час простою і зрозумілою.

Система застосовуєтбся в таких областях як:

– Функціональна діагностика.

– Клінічна кардіологія.

– Амбулаторні кардіологічні дослідження.

– Спортивна і спеціальна медицина.

– Наукові дослідження.

На даний момент написання БР, найдешевша модель системи моніторингу є «DIACARD» 2.1 (LE) З РЕЄСТРАТОРОМ 06000.7. ХОЛТЕР, яка коштуватиме 70000 гривень, тобто приблизно 1760 доларів. Це доволі справедлива ціна, як для професійного обладнання, але для невеликих медичних закладів це може стати вирішальним фактором, і тому потрібно вибирати більш дешевий варіант.

Наступним буде електрокардіограф від швейцарської компанії SCHILLER. Компанія SCHILLER була заснована 1974 року Альфредом Шиллером. Спочатку компанія виробляла кишенькові електрокардіографи для екстреного використання, а пізніше розпочато виробництво повноцінних багатоканальних апаратів ЕКГ, спірометрів і моніторів пацієнтів. Створена як індивідуальне підприємство, що спочатку розміщувалася в чотирикімнатній квартирі, компанія перетворилася на успішну корпорацію з 1400 співробітників, 30 філіалами та дилерською мережею по всьому світу. Сьогодні SCHILLER AG – це світовий лідер у виробництві та продажу приладів для

діагностики серцево-судинної системи, дефібриляції та моніторингу пацієнта, а також програмних рішень для медицини.

Розглянемо електрокардіограф CARDIOVIT AT-180. Ця багатофункціональна робоча станція поєднує в собі найбільший сенсорний екран і передові клінічні інструменти. Висококласний електрокардіограф SCHILLER AT-180 заснований на новій програмній платформі SCHILLER. Він пропонує надійну конструкцію для найбільш завантажених лікарень, а також поєднання додатків, що робить його ідеальною багатофункціональною робочою станцією для приватної практики.

Рисунок 2.2 – Електрокардіограф CARDIOVIT AT-180[15]

Рисунок 2.3 – Електрокардіограф CARDIOVIT AT-180 на спеціальній стійці[15]

Характеристики:

– 18,5-дюймовий мультисенсорний екран з високою роздільною здатністю для зручного перегляду ЕКГ.

– Повнорозмірна клавіатура з міцною змінною кришкою, що захищає від пилу, бруду і рідин, що робить її дуже гігієнічною.

– Ритм спокою до 20 хвилин.

– Каркас ЕКГ: створіть 10-секундну ЕКГ спокою із запису ритму спокою.

– Консультант із підключення з кольоровими хвильовими формами й анатомічною моделлю.

– Виявлення реверсу відведень.

– Швидкий і безпечний двонаправлений зв'язок через Wi-Fi, PDQ.

– Робочий список.

– Тест із фізичним навантаженням (опціонально).

– Різні програми діагностичного аналізу: ETM, ETM Sport, SAECG, Vector ECG, виявлення аритмії, повторний вимір ЕКГ (опціонально).

– Отримання та аналіз 16 відведень у стані спокою: на основі реальних даних вимірів, а не лише розрахунків (опція).

CARDIOVIT AT-180 професійно протестований на проникнення та кібербезпечний завдяки цим функціям:

– Управління доступом на основі ролей.

– Ідентифікація пацієнта за робочим списком/замовленнями, штрих-кодом, PDQ.

– Інтеграція служб каталогів через LDAP із протоколюванням аудиту.

– Зашифроване передавання даних.

– WPA2 корпоративний протокол WiFi.

– Налагоджено ядро Linux із підвищеним рівнем безпеки.

Компанія SCHILLER, починала з виробництва кишенькових електрокардіографів із перспективою розширення на багатоканальні апарати, спірометри і монітори пацієнтів. А зараз вона є глобальним лідером у виробництві та продажу приладів для діагностики серцево-судинної системи, дефібриляції та моніторингу пацієнта, а також програмних рішень для медицини.

Перейдемо від великих приладів до чогось більш меншого, наприклад до електрокардіографа з такою ж маленькою назвою як H12. Розвиток інтернету здійснив революцію в усіх галузях. Це торкнулося комп'ютерів, мобільних телефонів, годинників, окулярів і навіть медичних приладів. Концепція мініатюризації процвітає, і це передова тенденція, яку компанія Comen наслідує по стопах світу, щоб створити висококласний 12-канальний електрокардіограф H12. Компактний і вишуканий електрокардіограф має не тільки вирізнятися привабливим дизайном, а й чудовою продуктивністю. Він повинен поєднувати

в собі відмінний зовнішній вигляд і відмінне функціонування.

Компанія Comen, заснована в 2002 році в Шеньчжені, Китай, є виробником медичного обладнання з науково-дослідницьким потенціалом, що задовольняє потреби медичних працівників. Дотримуючись мислення, орієнтованого на кінцевого користувача, Comen прагне забезпечити найповнішу в світі лінійку продуктів моніторингу, перше в світі рішення для реанімації та інтенсивної терапії, першокласне рішення для операційних та інтенсивної терапії, керуючись концепцією «Спеціалізований продукт, спеціалізоване використання». Comen об'єднує зусилля з відомими експертами з найкращих вищих лікарень Китаю класу А для досягнення співпраці між лікарнями, науково-дослідними інститутами та підприємствами, перетворюючи інноваційні ідеї в продукти, що відповідають очікуванням користувачів. На сьогоднішній день 40% з майже 600 патентів компанії Comen отримані в результаті співпраці з експертами лікарень.

R&D – це серце високотехнологічної компанії. R&D центр Comen складається з 5 відділів: програмного забезпечення, апаратного забезпечення, алгоритмів, машин та промислового дизайну. У ньому працює понад 2 000 співробітників, 65% з яких мають ступінь магістра в галузі біомедицини. Comen наполягає на тому, що витрачає понад 20% свого річного доходу на R&D, щоб забезпечити швидкий запуск продуктів і постійну оптимізацію якості продукції.

Науково-дослідний персонал складає 33,3% від загальної кількості працівників компанії, і 80% з них походять з провідних університетів. Завдяки наполегливій праці, продукція Comen слугує більш ніж 50 000 медичних закладів та постачальників соціальних послуг у 130 країнах світу, обслуговуючи понад 300 мільйонів пацієнтів на рік.

Рисунок 2.4 – Електрокардіограф H12[16]

Характеристики:

– Ручне введення інформації про пацієнтів.

– Підтримка пошуку за ім'ям та ідентифікаційним номером пацієнта.

– Управління історіями пацієнтів допомагає легко шукати, передавати, друкувати і переглядати інформацію про пацієнтів.

– Підтримка режиму очікування та автоматичного пробудження, а також вимкнення за часом.

– Інтелектуальний режим заряджання. доступне як повільне, так і швидке заряджання. Режим заряджання синхронізується з робочим режимом для зменшення тепловиділення та захисту компонентів.

– Перегляд кривих ЕКГ у 12 відведень на 600 секунд.

– Внутрішня пам'ять на 10,000 ЕКГ. підтримка форматів DAT, BMP, JPG, PDF, DICOM, FDA- XML, SCP.

– Підтримка підключення до дротових/бездротових мереж.

– Підтримка підключення SD-карти, миші, клавіатури, сканера, лазерного принтера.

– Підтримка програмного забезпечення для управління даними ЕКГ, протокол HL7 / FTP / DICOM дає змогу підключатися до системи HIS, EMR, PACS лікарні. Лікар може переглядати звіт про діагноз і криві ЕКГ безпосередньо з комп'ютера.

Здавалось би, такий маленький пристрій, але який функціонал і зручність використання він дає. H12 – це ідеальний варіант в екстрених ситуаціях, коли поруч немає медичних закладів. В місцях, де є велике скупчення людей, таких як торгові центри, парки, зоопарки, великі міста і т.д. комусь може стати зле, втратити свідомість, отримати інфаркт. Тому, для таких ситуацій можна створити місця екстреної допомоги, де такий маленький та компактний кардіограф підходить якнайкраще.

Наступним нашим кандидатом буде портативна система ЕКГ Infinium QRS-12. Понад 30 років Infinium Medical розробляє та виготовляє монітори пацієнта та інші провідні технології для амбулаторної допомоги та високотехнологічних медичних установ.

Заснована у 1991 році, компанія Infinium Medical є визнаним розробником та виробником медичного обладнання, зокрема моніторів для пацієнтів. Її штаб-квартира розташована у США, і вона відома своєю репутацією у якості постачальника передових технологій у медичній галузі. Основна спеціалізація компанії - це розробка та виробництво моніторів для пацієнтів, а також обладнання для операційних та інтенсивної терапії. Крім американського ринку, Infinium Medical успішно працює на міжнародному рівні, маючи розгалужену міжнародну мережу продажів, що охоплює 87 країн.

Розроблена для швидкого темпу роботи, портативна система ЕКГ Infinium QRS-12 пропонує надзвичайно простий і адаптований сенсорний інтерфейс користувача. Інформація про пацієнта, а також параметри життєво важливих функцій можуть бути швидко змінені відповідно до мінливого стану

пацієнта. QRS-12 має 9-дюймовий сенсорний екран з високою роздільною здатністю та алфавітно-цифрову клавіатуру для оптимізації швидкості обслуговування пацієнта. Таким чином, користувач може проводити швидкий і зручний аналіз, маючи при цьому кілька варіантів друку і передачі даних.

На відміну від більшості апаратів ЕКГ, QRS-12 пропонує програмне забезпечення для інтерпретації в якості стандартної функції. QRS-12 має 2 "вбудовані" бази даних еталонних ЕКГ.

QRS-12 пропонує кілька рішень для підключення до мережі та управління даними пацієнта на платформі електронних медичних записів або в інформаційній системі HL7. ЕКГ система пропонує Ethernet і RS232 з'єднання з відкритим вихідним кодом протоколу зв'язку. Infinium пропонує 2 рівні мережевих підключень. QRS-12 сумісний з HL7 і DICOM. Мережевий протокол HL7 дозволяє передавати і зберігати всю інформацію про пацієнта і тенденції життєво важливих показників в лікарняній інформаційній системі. Для медичних установ, які не підтримують протокол HL7, існує програмне забезпечення Infinium ECG manager , яке дозволяє передавати всі збережені файли ЕКГ і дані пацієнта на віддалений ПК. Файли пацієнтів можна дуже просто зберігати, роздруковувати і передавати.

Рисунок 2.5 – Портативна система ЕКГ Infinium QRS-12[17]

Характеристики:

– Розумний і портативний дизайн.

– 9-дюймовий сенсорний TFT-екран високої роздільної здатності з регульованим кутом нахилу, зручний для перегляду.

– Водонепроникна і повна алфавітно-цифрова клавіатура з клавішами швидкого доступу.

– Широка частотна характеристика 0,05 ~ 250 Гц для розпізнавання та захоплення слабких сигналів, що більше підходить для педіатрії.

– Специфічна вікова та гендерна диференціальна валідність в програмі аналізу CardioPro максимізує точність інтерпретації ЕКГ.

– Унікальний модуль збору ЕКГ з 3 незалежними ключами спрощує весь робочий процес.

– Схема розміщення електродів ЕКГ для керівництва по підключенню

відведень і сигналізації.

– Перегляд і запис заморожених форм хвиль тривалістю до 300 секунд.

– До 300 секунд R-R аналіз в 1 або 3 ритмах для полегшення локалізації аритмії.

– Доступний режим Cabrera.

– Підтримка різних форматів файлів: XML, DICOM, JPG і PDF.

– Локальне сховище для зберігання до 1500 файлів.

– Способи зберігання файлів: локальна пам'ять, SD-карта, USB-накопичувач.

– Програмне забезпечення для управління ЕКГ через LAN або WIFI (опціонально).

Портативна ЕКГ-система Infinium QRS-12-це передове рішення в області медицини, що забезпечує зручний і точний моніторинг стану пацієнта. Її функціональність і надійність дозволяють з упевненістю використовувати її як в медичних установах, так і амбулаторно. Infinium Medical відома своєю високою якістю та інноваційним підходом до медичних технологій. Їх продукція широко використовується як на внутрішньому ринку США, так і в Міжнародному медичному співтоваристві, демонструючи їх впевнену позицію в галузі.

І останій в нашому списку буде ручний портативний монітор серцевого ритму ЕКГ PC-80B[18][19]. Призначений для формування і запису сигналу ЕКГ і середньої частоти серцевих скорочень дорослого пацієнта. Він застосовний для використання в клініках і будинках, і зручний для роботи самими пацієнтами. Його можна використовувати для безперервного вимірювання за допомогою клейових накладок ЕКГ і свинцевого дроту, під'єднаного до приладу.

Пристрій не є монітором ЕКГ, як використовується в клінічній установі або лікарні, але використовується тільки для вибіркової перевірки. Він може бути використаний для заміни звичайного обстеження ЕКГ або моніторингу в

режимі реального часу.

Результати вимірювань є корисним орієнтиром для лікаря, але не приймають прямого діагностичного або аналітичного рішення на основі інформації, наданої цим пристроєм.

Форма хвилі ЕКГ та інтерпретація результатів чітко відображаються на РК-екрані. Передова вимірювальна технологія забезпечує стійкі та точні форми сигналів ЕКГ. Може бути представлено сімнадцять типів результатів вимірювань. До 1200 записів для швидкого вимірювання або 10-годинних записів даних для безперервного вимірювання можуть зберігатися у вбудованій пам'яті. Записи даних також можна переглянути, скопіювати, видалити та завантажити на комп'ютер за допомогою кабелю для передачі даних. Енергозберігаюча техніка з функцією автоматичного вимкнення. Дві батарейки типу AAA використовуються для підтримки понад 10 годин робочого часу.

Рисунок 2.6 – Ручний портативний монітор серцевого ритму ЕКГ PC-80B[52]

Характеристики:

– Одноканальний сигнал ЕКГ.

– Вбудовані металеві електроди або 3 клейкі прокладки ЕКГ, з'єднані з роз'ємом приладу зі свинцевим проводом.

– Пропускна спроможність ЕКГ: 1Гц-40Гц.

– Рівень внутрішнього шуму: ≤30уВп-п.

– Діапазон частоти серцевих скорочень: 30-240 об/хв.

– Точність серцевого ритму: ±2 об/хв або ±2%, залежно від того, що більше.

– Коефіцієнт відбраковування в загальному режимі (КОСС): ≥60дБ.

– Швидкість розмаху хвилі: 20мм/с±10%.

Можна сказати, що РС-80В – це ідеальний варіант для домашніх обстежень. Простий спосіб слідкувати за здоров’ям свого серця та вчасно реагувати на ситуації, які б могли призвести до летальних наслідків. Було б чудово, якби такий прилад був у кожному домі, тоді шанс на вчасно врятоване життя піднявся би трохи вище.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для цьогу проєкту було обрано мікроконтролер ESP32 DEVKITV1. Оригінальний Arduino Uno був випущений понад десять років тому, у 2010 році, і досі залишається однією з найпопулярніших плат для проєктів завдяки великій документації та підтримці. Однак останнім часом з'явилося нове покоління мікроконтролерів з підвищеною продуктивністю і можливостями підключення. Ці нові мікроконтролери дають нам змогу створювати більш просунуті проєкти з інтегрованим опрацюванням відео та потокового передавання даних у мініатюрному корпусі з низьким енергоспоживанням, що

робить їх ідеальними для недорогих IoT-пристроїв. Наразі чемпіоном серед цих мікроконтролерів з можливістю підключення є серія ESP32.

ESP32 – це серія недорогих і малопотужних чіпів, розроблених компанією Espressif Systems, які є потужнішим наступником ESP8266. ESP32 був випущений у 2016 році і має двоядерний дизайн з підтримкою WiFi і дворежимного Bluetooth. Мікроконтролер оснащений 32-бітовим мікропроцесором Tensilica Xtensa LX6, що працює на частоті 240 МГц, і співпроцесором з ультранизьким енергоспоживанням.

Мікромодуль ESP-WROOM-32, що використовується в 30-контактному модулі розробника DEVKITV1, є новим високопродуктивним модулем Wi-Fi + BT + BLE від Espressif. Розроблений для широкого спектру застосувань, від простих мережевих датчиків до більш складних завдань, таких як обробка звуку, передача звуку і кодування MP3, модуль DEVKITV1 містить всю периферію, необхідну для швидкого запуску ESP-WROOM-32. Однією з особливостей модуля є його дуже низьке енергоспоживання, яке може становити всього 20 мкА з можливістю вибору різних режимів сну.

Рисунок 2.7 – ESP32 DEVKITV1 видом з переду[53]

Рисунок 2.8 – ESP32 DEVKITV1 видом з ззаду[53]

Цей модуль має повну інтеграцію мінімальної периферії, що забезпечує простоту в підключенні. Програмування модуля здійснюється шляхом простого підключення через USB. Він підтримує двомодовий Bluetooth - як "класичний", так і BLE. Швидкість Wi-Fi досягає 150 Мбіт/сек у стандартах 802.11 b/g/n, при цьому модуль підтримує різні режими Wi-Fi, такі як клієнт, точка доступу, Sniffer та Wi-Fi Direct. Модуль працює в широкому діапазоні температур від -40°C до +125°C. У режимі глибокого сну його енергоспоживання становить до 20 мікроампер. ПЗ може бути оновлено безпосередньо через повітря, а також існує можливість підключення до 4 x 16 МБ зовнішньої QSPI Flash і SRAM.

Модуль використовує USB-UART конвертер CP2102 і працює при напрузі живлення 5 В з максимальним струмом стабілізатора напруги 800 мА. Він відповідає стандартам FCC/CE/IC/TELEC/KCC/SRRC/NCC для Wi-Fi і підтримує протоколи 802.11 b/g/n/d/e/i/k/r (з швидкістю 802.11n до 150 Мбіт/с). Модуль підтримує A-MPDU і A-MSDU, а також захисний інтервал в 0.4 секунди. Він оснащений Flash-пам'яттю об'ємом 448 КБ, зовнішньою Flash-пам'яттю 4 МБ і SRAM-пам'яттю 520 КБ. Частотний діапазон модуля складає

2.4 ~ 2.5 ГГц, і він підтримує Bluetooth v4.2 BR/EDR і BLE specification. Має радіо NZIF приймач з чутливістю -98 dBm і передавач класу 1, класу 2 і класу 3 AFH. Для аудіо використовуються формати CVSD і SBC. Щодо апаратних засобів та інтерфейсів, модуль має різноманітні можливості, такі як SD, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR. Також є GPIO, сенсорний датчик, ADC, DAC, LNA передпідсилювач. На борту модуля розташовані Hall sensor і температурний датчик. Його генератори включають кварцовий на частоті 26 МГц і 32 кГц. Мікромодуль працює в діапазоні напруги від 2.2 до 3.6 В, середній робочий струм складає 80 мА, а піковий - 500 мА. Діапазон робочих температур від -40°C до +85°C. Відстань між контактними пінами становить 25.5 мм, а розміри плати - 27x51 мм.

Це програмне забезпечення підтримує різні режими Wi-Fi, включаючи Station, softAP, SoftAP+station і P2P. Щодо захисту, воно пропонує різні опції, такі як WPA, WPA2, WPA2-Enterprise і WPS. Шифрування забезпечується за допомогою таких протоколів, як AES, RSA, ECC і SHA. Оновлення програмного забезпечення можливе через UART, по мережі (OTA) або завантаження та запис програми через хост. Для розробки власного програмного забезпечення доступні різні інструменти, включаючи розробку на хмарних серверах і використання SDK. Щодо мережевих протоколів, підтримуються IPv4, IPv6, SSL, а також TCP, UDP, HTTP, FTP і MQTT. Користувач може налаштовувати параметри за допомогою інструкцій AT, хмарного сервера або застосунків для Android/iOS.

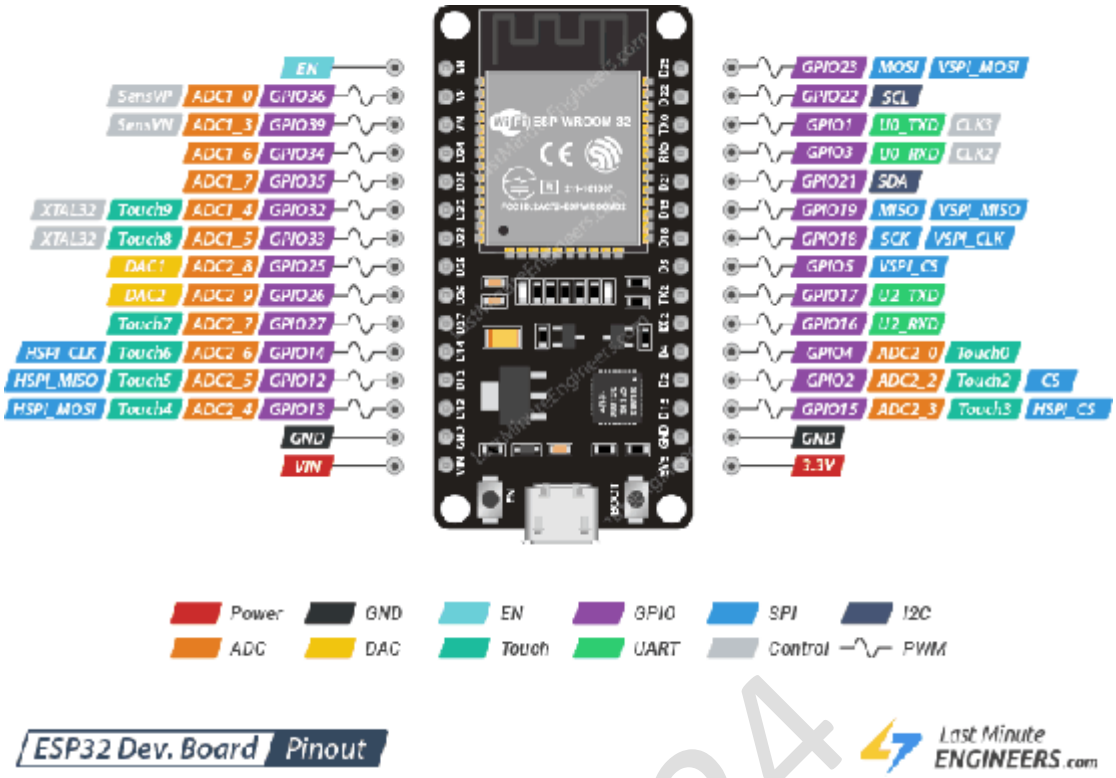

Рисунок 2.9 – Функціональне призначення пінів ESP32 DEVKITV1[54]

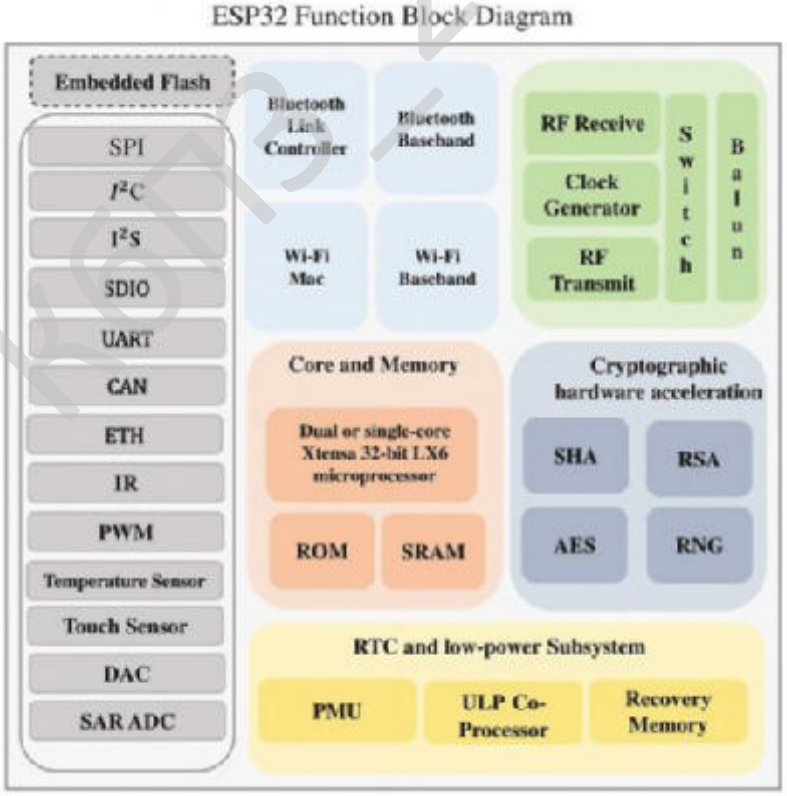

Рисунок 2.10 – Структурна схема внутрішніх компонентів мікроконтролера[55]

AD8232 – це вбудований модуль для створення сигналу, призначений для вимірювання ЕКГ та інших показників. Він призначений для вилучення, підсилення та фільтрації малих сигналів біопотенціалу в умовах шуму, наприклад, викликаного рухом або віддаленим розміщенням електродів. Ця архітектура спрощує отримання вихідного сигналу за допомогою вбудованого мікроконтролера та аналого-цифрового перетворювача(АЦП) з дуже низьким рівнем енергоспоживання.

AD8232 використовує біполярний фільтр високих частот для усунення артефактів руху і потенціалу напівелементів електродів. Цей фільтр є невід'ємною частиною приладової архітектури підсилювача і дозволяє об'єднати в одному блоці високий коефіцієнт підсилення і фільтрацію високих частот, що економить місце і знижує вартість.

Для подальшого прибирання шумів можна використовувати операційний підсилювач для створення 3-полюсного фільтра нижніх частот. Користувач може налаштувати частоту зрізу кожного фільтра за потребою, щоб його можна було використовувати для різних застосувань. Для поліпшення синфазного відсікання лінійних частот в системі та інших небажаних перешкод, містить підсилювач для додатків з керованим виводом, таких як привід правої ноги (RLD).

AD8232 має функцію швидкого відновлення, яка зменшує тривалість довгого періоду затухання фільтрів високих частот. Після різкої зміни сигналу, що виводить підсилювач з ладу (наприклад, відключення виводів), AD8232 автоматично налаштовується на більш високу частоту зрізу фільтра. Ця функція дозволяє швидко відновлюватися, а отже, проводити достовірні вимірювання незабаром після підключення електродів до пацієнта.

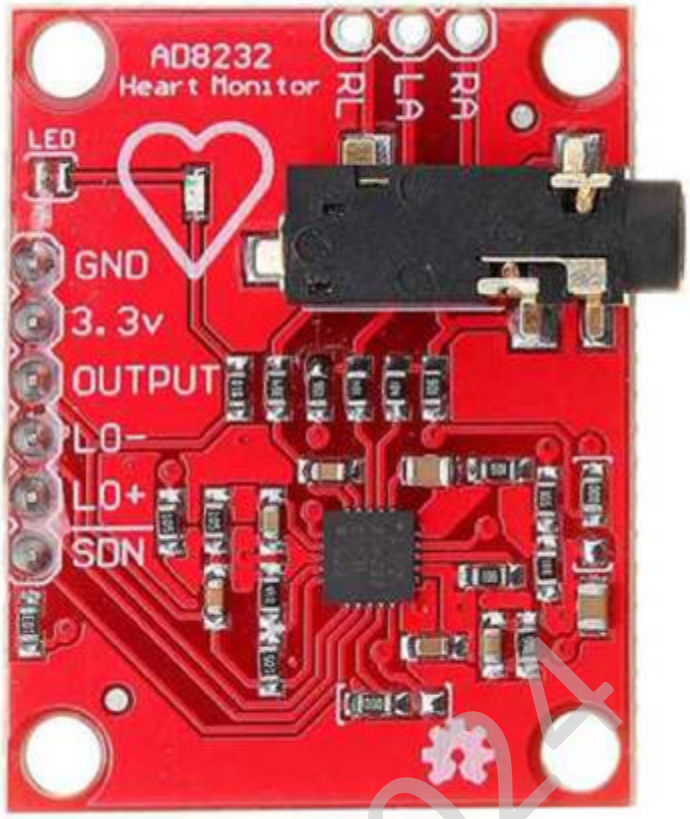

Рисунок 2.11 – Датчик ЕКГ AD8232[56]

Рисунок 2.12– Електроди датчика AD8232[57]

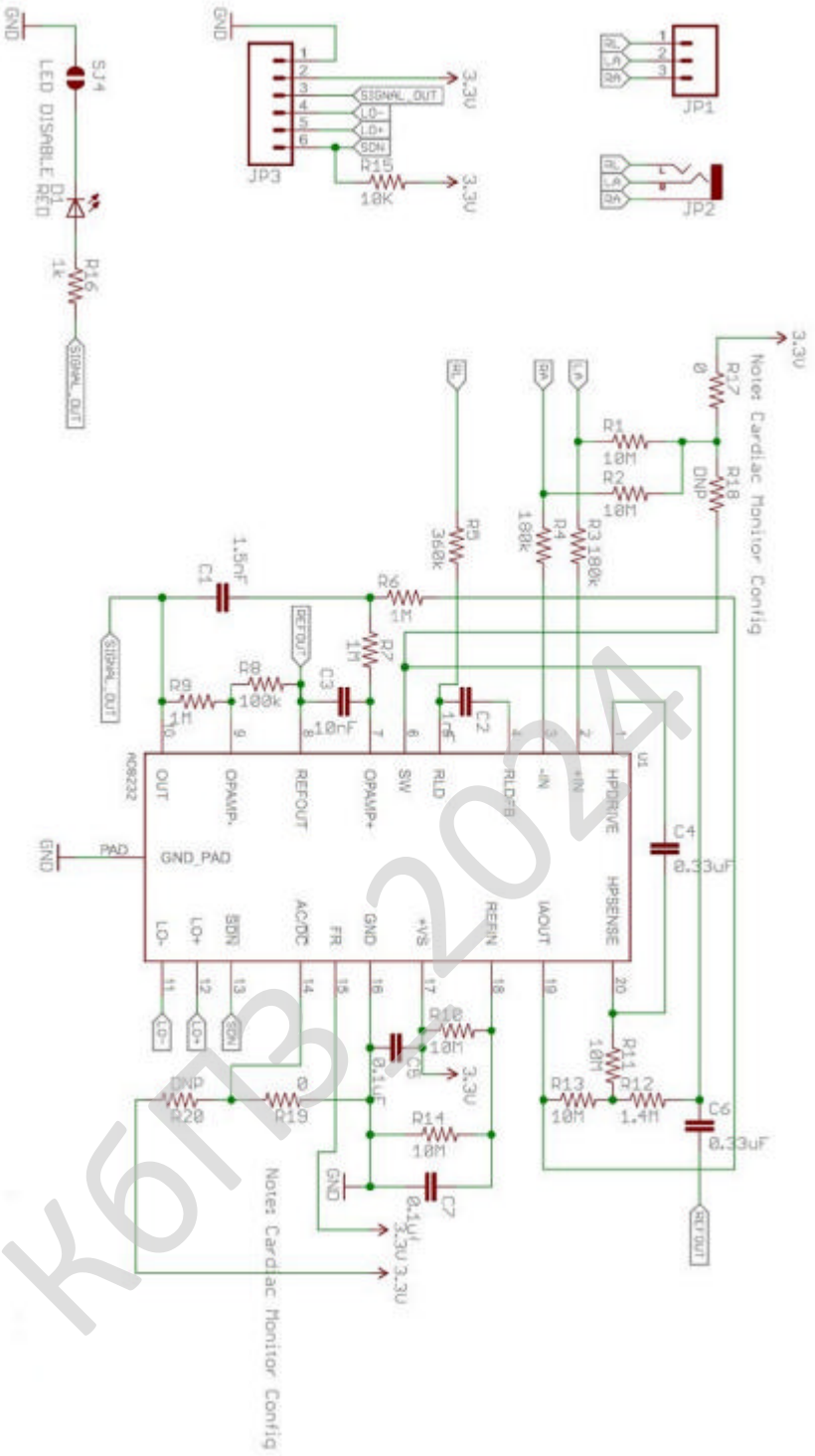

Рисунок 2.13 – Електрична схема датчика AD8232[12]

Фронтальний блок для одновідвідної ЕКГ повністю інтегрований і має низький струм живлення, що складає 170 мкА (типовий). Він забезпечує високу якість сигналу, з коефіцієнтом відхилення синфазного сигналу 80 дБ від постійного струму до 60 Гц. Цей блок може працювати в конфігураціях з двома

або трьома електродами, а також має вбудований операційний підсилювач без фіксації. Для кращої фільтрації сигналу він має 2-полюсний регульований фільтр високих частот і 3-полюсний регульований фільтр нижніх частот із регульованим коефіцієнтом посилення. Цей блок також має функцію швидкого відновлення покращення налаштувань фільтра і вбудований детектор вимкнення виводів змінного або постійного струму. Щодо живлення, він працює в діапазоні від 2,0 В до 3,5 В і має вихід Rail-to-rail. Внутрішній фільтр радіоперешкод забезпечує стабільну роботу в умовах впливу радіочастотних сигналів, а контакт вимкнення забезпечує додатковий функціонал для управління пристроєм.

Модуль живлення MB102 3.3V/5V використовується для стабілізації вхідної напруги від зовнішніх джерел живлення та подачі її на плати. За замовчуванням модуль використовується для подачі стабілізованої напруги на макетні плати. Можливе підключення до двох макетних плат.

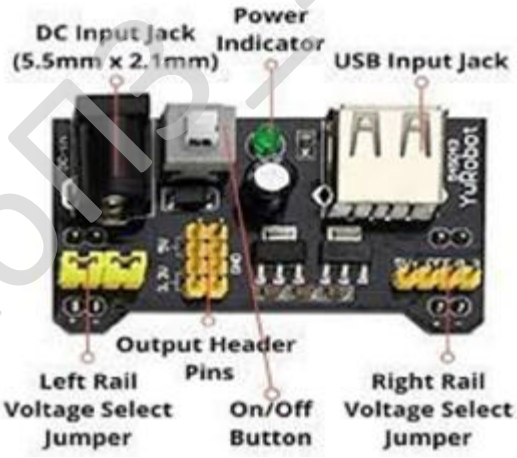

Рисунок 2.14 – Модуль живлення MB102[58]

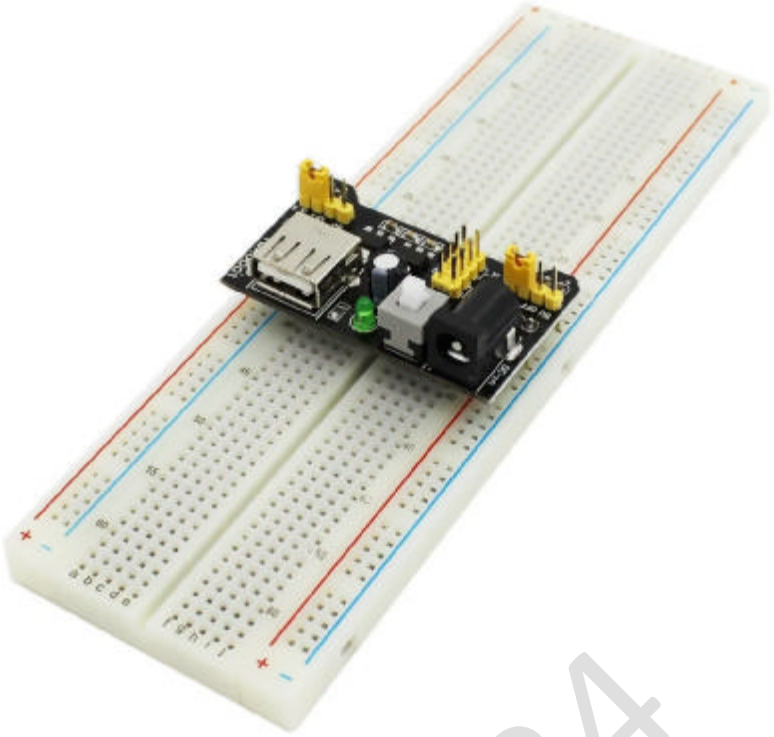

Рисунок 2.15 – Макетна плата та модуль живлення MB102 разом[59]

Характеристики:

– Є перемикач ON-OFF.

– Є вхід USB (Type-A).

– Доступний вхід DC Barrel Jack.

– Вбудований світлодіодний індикатор живлення.

– Два незалежні канали.

Технічні характеристики:

– Модуль живлення для макетної плати, сумісний з 5, 3,3 В.

– Застосовується для живлення макетних плат MB102.

– Вхідна напруга: 6.5-12 В (DC) або джерело живлення USB.

– Вихідна напруга: 3.3V/5V може перемикатися.

– Максимальний вихідний струм: <700 мА.

– Незалежний контроль коливань двох доріг може переключатися на 0 В, 3,3 В, 5 В.

Для даного проєкту була обрана мова C/C++, тому що вона сумісна з

мікроконтролером. ESP32 DEVKITV1 підтримує мову програмування C/C++, яка є стандартом для вбудованих систем, включаючи мікроконтролери. Завдяки цьому, є можливість працювати з апаратурними ресурсами на низькому рівні, що дозволяє оптимізувати роботу з ESP32 та отримувати високу ефективність програм.

Середовищем розробки стала Arduino IDE. Arduino IDE – це ПЗ, призначене для створення власних програм для платформи Arduino, а також для інших мікроконтролерів. Це популярне середовище розробки, особливо корисне для творчих конструкторів, що використовують Arduino для створення простих систем автоматизації та робототехніки. Середовище надає велику кількість вбудованих бібліотек, що полегшують розробку програм. Їх можна знайти в офіційному репозиторії Arduino IDE, а також на спеціалізованих веб-сайтах та форумах, де користувачі діляться своїми власними розробками та бібліотеками для використання в проектах. Одним з найбільших таких сайтів є Github, на якому є майже всі бібліотеки які тільки треба для будь-яких проєктів, також бібліотеки можна знайти на сайтах, де продають датчики, модулі і тд.

Враховуючи вищесказане, мова програмування C/C++ та Arduion IDE – це ідеальний варіант для реалізації будь-яких проєктів які тільки можна придумати.

2.3 Розгорнута постановка завдання

В процесі розробки БР необхідно виконати наступний обсяг робіт:

– Обґрунтувати актуальність теми БР.

– Визначити призначення системи та область її застосування.

– Переглянути аналогічні існуючі системи з метою виявлення їх переваг та недоліків.

– Обґрунтування вибору засобів для побудови системи, а саме таких як: мікроконтролер, датчик ЕКГ та модуль живлення, а також мови програмування.

– Написання програми для зчитування даних з датчика, обробки і аналізу цих даних, а також відображення результатів на спеціальному веб-сайті в реальному часі.

– Протестувати систему, щоб виявити та усунути можливі помилки та недоліки.

– Написати перелік умовних позначень, символів, одиниць та термінів, що будуть використані при написанні БР.

– Надати список літератури, що використалась при написанні БР.

– Написати анотацію на українській та англійській мовах.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Темою БР була розробка ПЗ для системи моніторингу ЕКГ людини.

Проаналізувавши існуючі аналогові системи та визначивши їх переваги, було прийнято рішення розробити ПЗ з виявленими перевагами та без недоліків для вирішення проблем, пов'язаних з технічними труднощами.

Основна функція даної системи – отримання, обробка та зберігання ЕКГ. Кожному цьому пункту відповідає певна частина системи. Тому, основними критеріями функціонування системи має бути:

– Підключення необхідних бібліотек для датчика ЕКГ та мікроконтролера.

– Налаштування пінів підключення для кожного компонента.

– Ініціалізація зв'язку з датчиком ЕКГ та мікроконтролером.

– Запуск зчитування значень електричних імпульсів серця з датчика ЕКГ.

– Перевірка успішності зчитування даних.

– Збереження значень електричних імпульсів серця для подальшого використання.

– Передача по Wi-Fi даних.

– Оновлення сторінки веб-сайту для відображення останніх результатів.

– Перевірка та аналіз отриманих даних з датчика ЕКГ.

– Завершення поточного циклу.

– Вимкнення модуля живлення та повторення алгоритму для продовження моніторингу.

Цей алгоритм слугує загальним каркасом роботи системи ЕКГ, де вимірюються електричні імпульси серця, а отримані дані виводяться на

відповідному веб-сайті для подальшого аналізу та моніторингу стану серцево-судинної системи.

Враховуючи обмежений бюджет, важливо, щоб система була економічно ефективна та доступна. Тому, однією з головних вимог до розробки цього приладу є вартість розробки. Із аналізу аналогічних систем, зрозуміло, що на ринку є багато досить дорогих електрокардіографів, які професійно використовуються в різних сферах. Варто зазначити, що багато модулів та датчиків є дорогими. Тому при розробці були використані не дуже дорогі компоненти, що дозволило знизити загальну вартість проєкту. Також використання ESP32 DEVKITV1 як основи для розробки системи дозволяє легко розширювати функціонал і замінювати компоненти за необхідності. Розробка цього пристрою власними силами не тільки зменшує витрати, але й надає можливість налаштувати систему під конкретні потреби користувача.

Вибір датчика не був складним, оскільки існує не так багато датчиків, придатних для системи ЕКГ моніторингу. Найкращим варіантом виявився датчик ЕКГ AD8232, але система повинна бути масштабованою, щоб за потреби можна було розширювати функції або додавати нові компоненти. Тому, якщо розглядати систему як систему моніторингу стану здоров'я пацієнта, то можливим напрямком розвитку системи є додавання датчиків, які вимірюють інші фізичні параметри. Це дозволило б отримати більш повну картину стану здоров'я пацієнта і надати додаткову інформацію для моніторингу та діагностики. Однак це значно підніме ціну системи моніторингу, оскільки кожен новий датчик потребує нових витрат, що зробить систему не дуже доступною для пересічного користувача. Але, якщо гроші не проблема, то можна розглянути наступні додаткові датчики:

– Датчики пульсу та оксиметрія: використання пульсоксиметра, наприклад, моделі MAX30102 або MAX30105, дозволить виміряти пульс та рівень кисню в крові. Це значно розширить можливості системи і надасть додаткову інформацію про серцевий ритм та сатурацію киснем.

– Датчики температури: підключення цифрових температурних датчиків, таких як DS18B20 або DHT11/DHT22, дозволить виміряти температуру повітря або тіла. Цей параметр може бути корисним для виявлення запалень, інфекцій або гіпертермії.

– Датчики активності та руху: інтеграція акселерометра, наприклад, моделі MPU6050 або LIS3DH, та гіроскопа, такого як L3G4200D або MPU6050, дозволить виміряти рухи та активність особи. Це дозволить відстежувати рівень фізичної активності.

– Датчики дихання: включення датчиків дихання на основі термопари або п'єзоелектричних датчиків дозволить виміряти ритм та глибину дихання. Це стане корисним для виявлення порушень дихання.

– Датчики глюкози в крові: додавання електрохімічних сенсорів глюкози, таких як FreeStyle Libre або Dexcom G6, дозволить вимірювати рівень глюкози в крові. Це особливо корисно для пацієнтів з цукровим діабетом та допоможе контролювати рівень цукру в крові в режимі реального часу.

– Датчики артеріального тиску: інтеграція датчиків, наприклад, моделі BMP280 або MPX5010, дозволить виміряти артеріальний тиск. Це може бути корисно для виявлення гіпертензії та інших серцево-судинних захворювань.

– Датчики рівня карбону діоксиду (CO2): включення CO2-датчиків, таких як MH-Z19 або SCD30, дозволить вимірювати рівень вуглекислого газу в атмосфері. Це може бути корисно для оцінки якості повітря та виявлення гіпоксії.

– Датчики електродермальної активності (EDA): інтеграція датчиків, наприклад, моделі GSR-10 або Grove - GSR, дозволить виміряти рівень потовиділення, що може бути показником стресу або емоційного стану.

– Датчики співвідношення кардіореспіраторної варіабельності (HRV): включення датчиків, які вимірюють HRV, дозволить оцінити зміни у серцевому ритмі, що може бути корисним для аналізу стресових реакцій та загального стану пацієнта.

Інтеграція цих додаткових датчиків в систему підвищить її функціональність, надасть більш повну інформацію про стан здоров'я та дозволить виявляти різноманітні патологічні стани. Врахування технічних можливостей мікроконтролера ESP32 DEVKITV1 та вибір підходящих датчиків буде важливим етапом для успішної реалізації розширеної системи моніторингу. Такі покращення дозволять забезпечити точнішу і детальнішу інформацію для аналізу даних, покращити точність діагностики та надати користувачам більше інформації щодо їх стану здоров'я.

3.2 Розробка структурної схеми

В даному розділі буде наведено детальну структурну схему системи ЕКГ моніторингу. Вона показує внутрішню будову та взаємозв'язки між основними компонентами приладу. Ця схема допомагає розуміти, як працює пристрій, які функції виконують його окремі частини, і як вони взаємодіють одна з одною для досягнення певної мети.

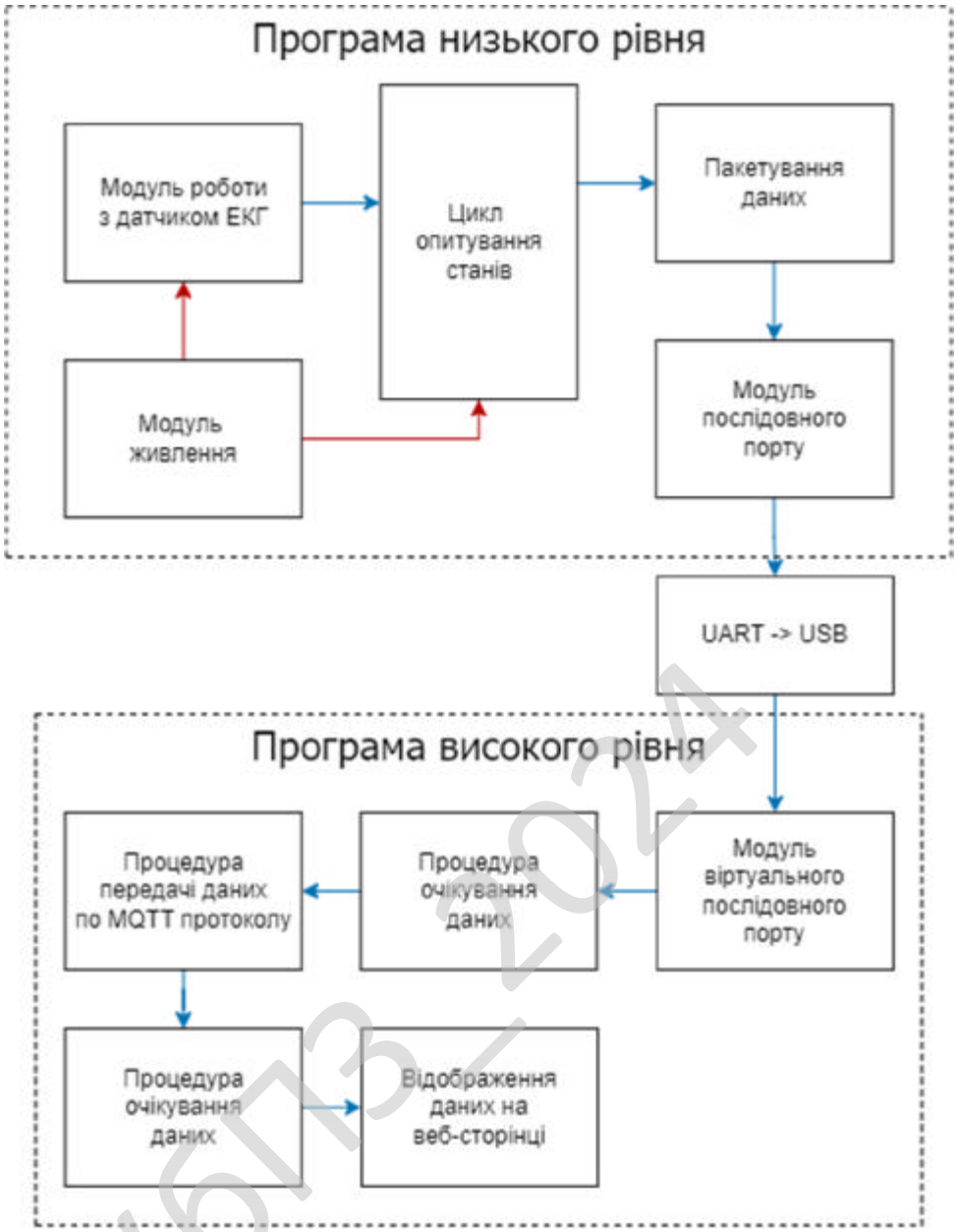

Рисунок 3.1 – Структурна схема пристрою

Дана структурна схема показує всі взаємозв'яки між компонентами системи. В основі системи лежить мікроконтролер, який обробляє дані від датчика ЕКГ. Після обробки, дані відправляються по протоколу MQTT на веб-сайт.

3.3 Розробка функціональної схеми

В даному розділі буде наведено детальну функціональну схему системи ЕКГ моніторингу. Вона показує функціональні блоки приладу та взаємозв'язки між ними. На відміну від структурної схеми, яка детально показує внутрішню будову приладу, функціональна схема концентрується на функціях та зв'язках між різними частинами пристрою.

Ця схема допомагає зрозуміти, які завдання виконує прилад у цілому, які функції відповідають кожному блоку, та як ці блоки взаємодіють для досягнення загальної мети пристрою.

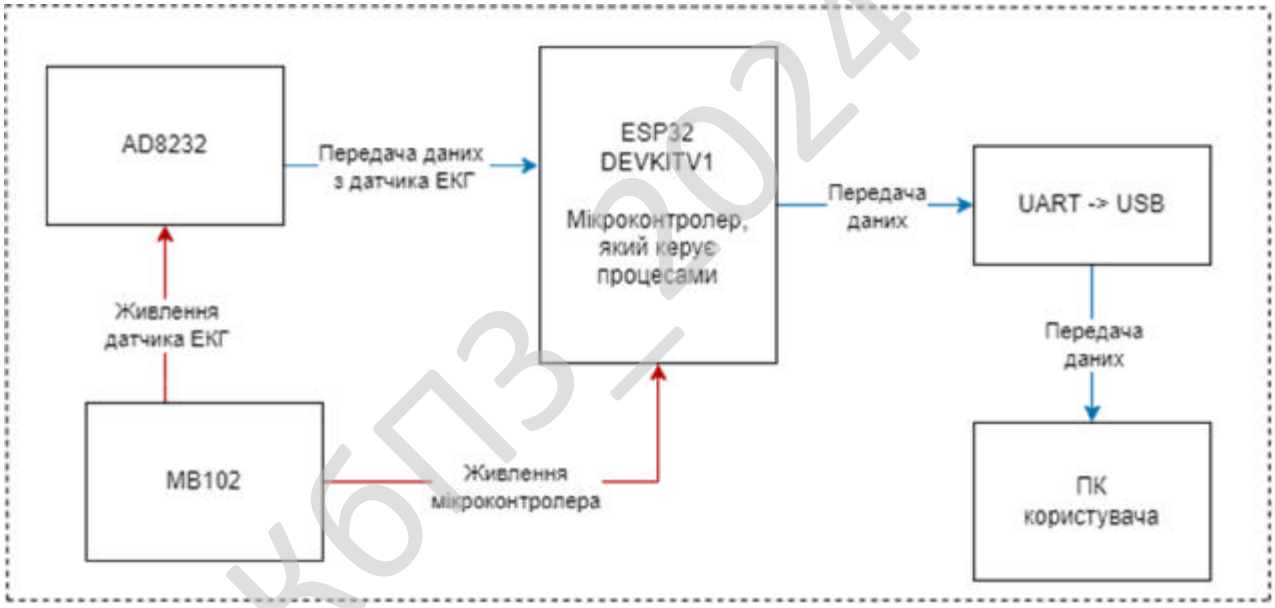

Рисунок 3.2 – Функціональна схема пристрою

Дана функціональна схема показує роль кожного компоненту впродовж всьої роботи пристрою. Розглядаються принципи взаємодії між компонентами системи, які спрямовані на забезпечення їхньої ефективної роботи та надійності, підкреслюється важливість їхньої внутрішньої взаємодії та спільної дії. Також у цьому розділі висвітлені аспекти, які стосуються обміну даними між компонентами системи, обробки та аналізу отриманих вимірювань.

3.4 Розробка діаграми процесів

Схема містить лише 3 компоненти, які забезпечують функціональність системи ЕКГ. Кожен з цих компонентів надійно підключений до мікроконтролера ESP32 DEVKITV1 через макетну плату. Принцип підключення заснований на використанні цифрових і аналогових входів і виходів ESP32.

Кожен компонент схеми підключається до відповідного порту на ESP32. Використання макетної плати спрощує розміщення і підключення компонентів, забезпечує надійність з'єднань і полегшує налагодження і модифікацію пристрою.

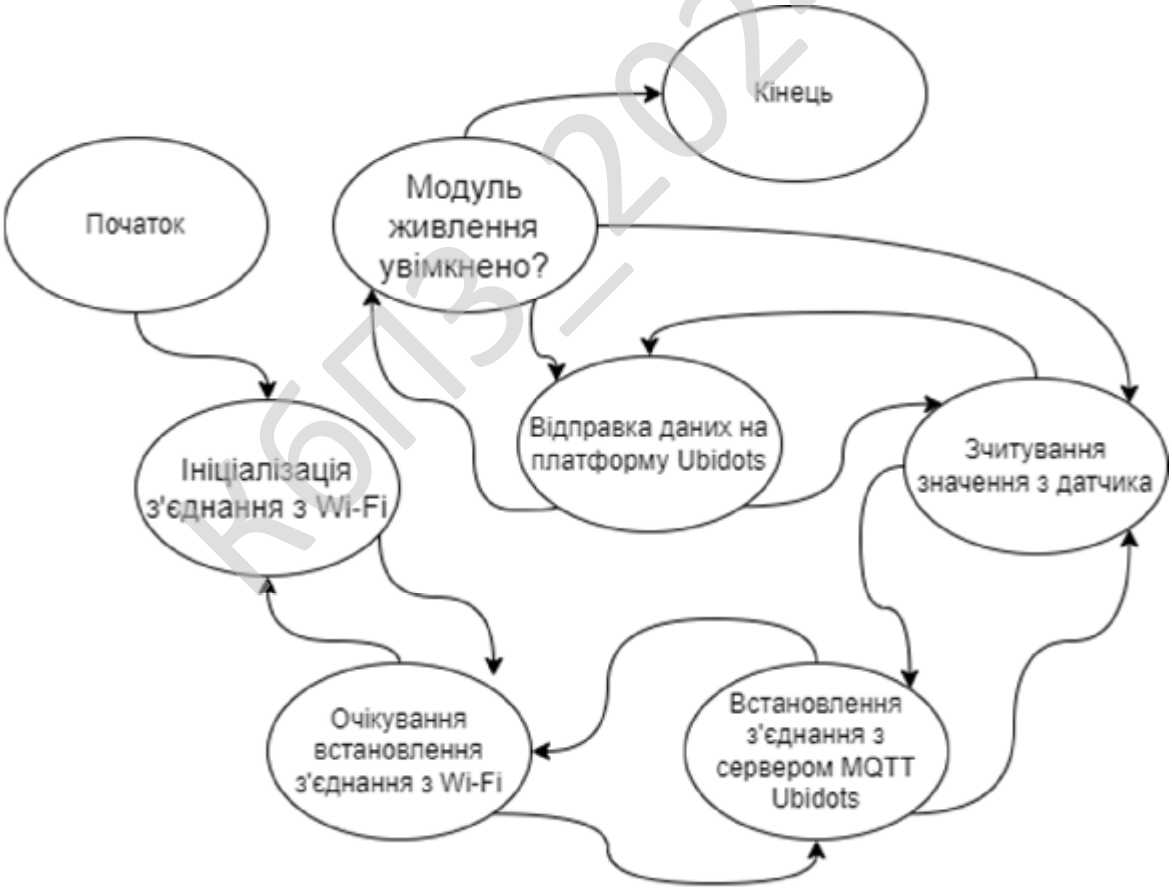

Рисунок 3.3 – Діаграма взаємодії процесів

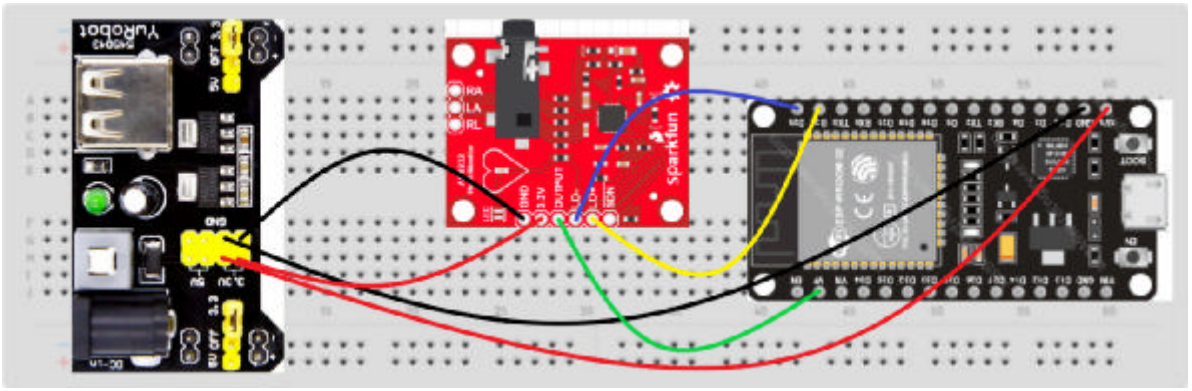

Рисунок 3.4 – Схема системи ЕКГ моніторингу

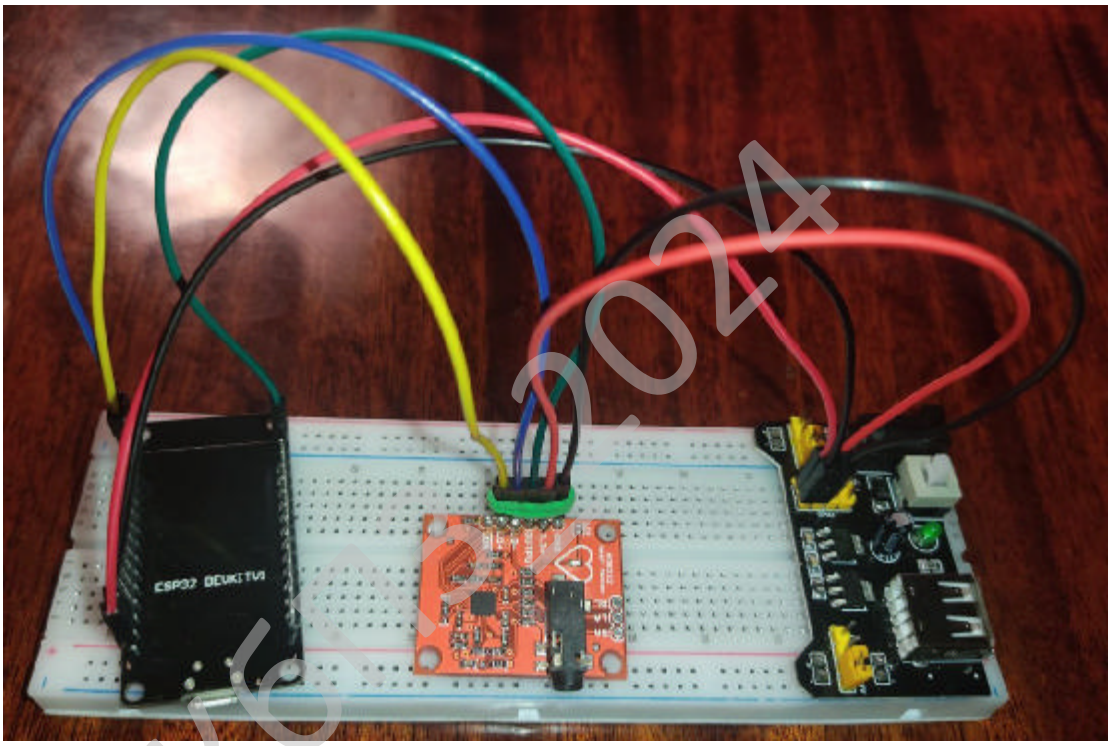

Рисунок 3.5 – Схема системи ЕКГ моніторингу на реальному фото

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

В цьому розділі наведено алгоритм роботи системи та її блок-схема. Розписані всі необхідні кроки для розуміння того, як працює система ЕКГ та які її функції.

Поетапний алгоритм роботи системи ЕКГ моніторингу представлений нижче:

Крок 1: Початок програми.

Крок 2: Ініціалізація з'єднання з Wi-Fi.

Крок 3: Очікування встановлення з'єднання з Wi-Fi.

Крок 4: Встановлення з'єднання з сервером MQTT Ubidots.

Крок 5: Перевірка з'єднання з сервером MQTT.

Крок 6: Зчитування значення з датчика.

Крок 7: Відправка даних на платформу Ubidots.

Крок 8: Обробка вхідних та вихідних повідомлень MQTT.

Крок 9: Якщо живлення присутнє, то повторення циклу.

Крок 10: Якщо живлення відсутнє, то завершення циклу.

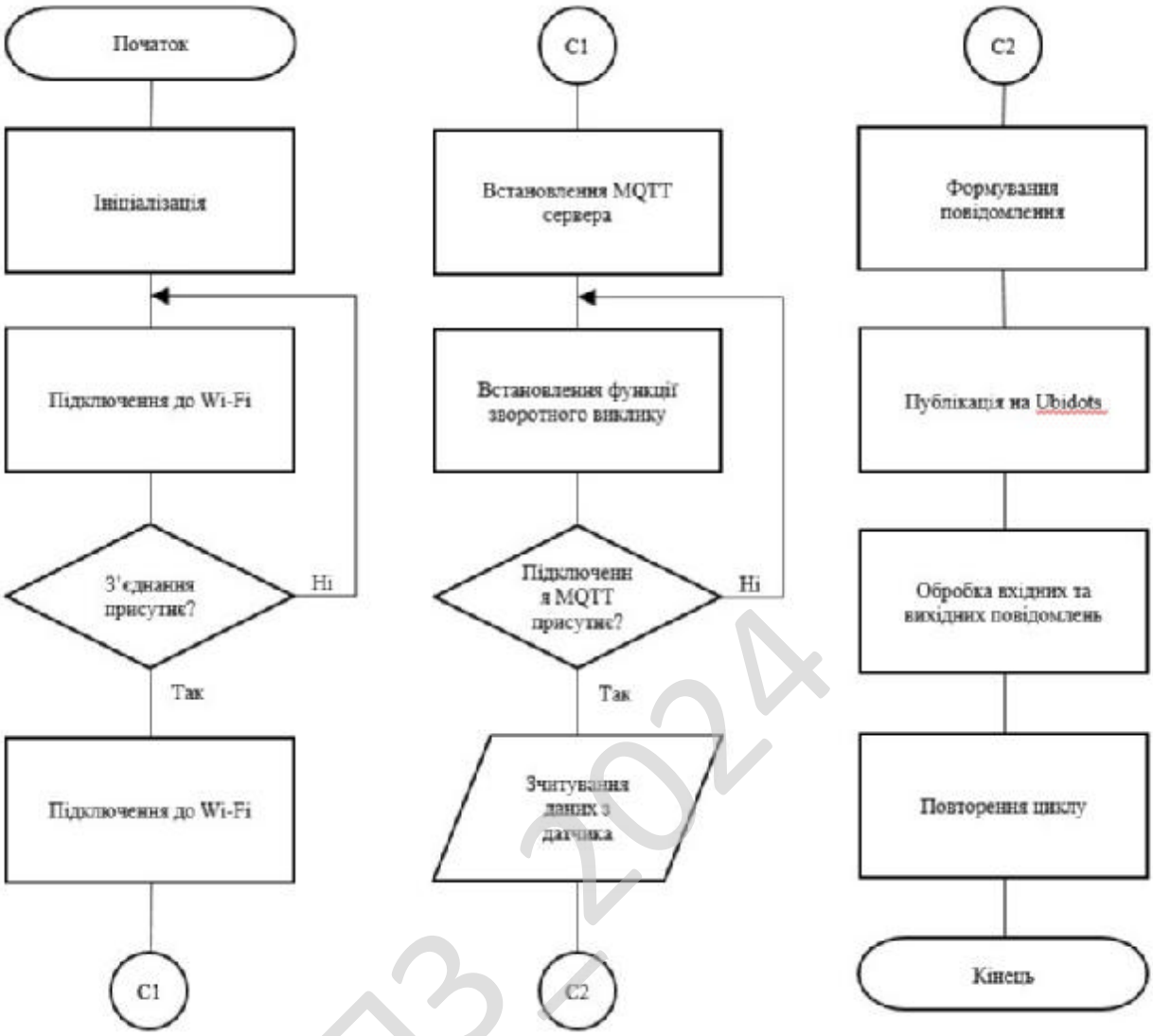

Рисунок 4.1 – Блок-схема основної програми

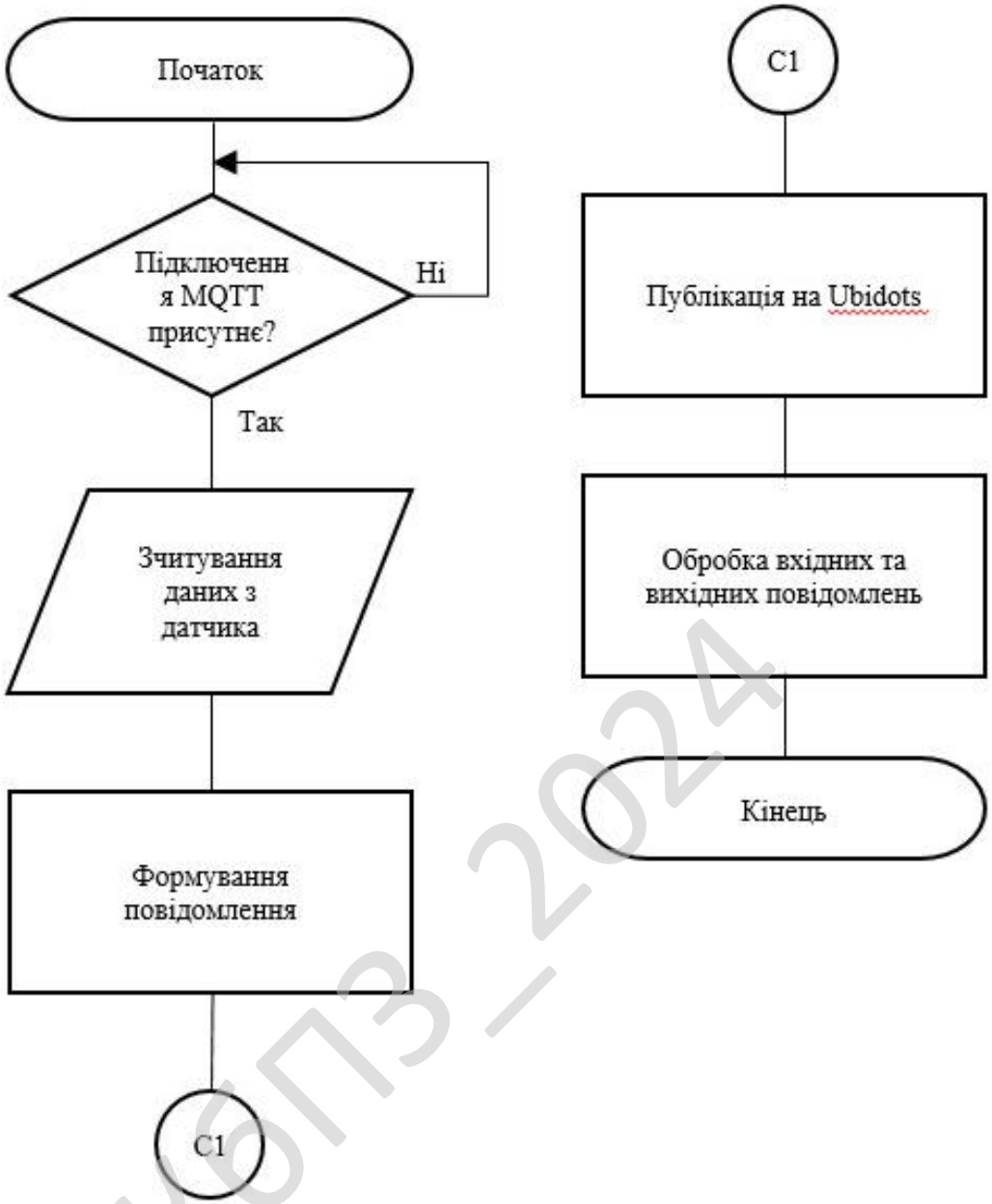

Рисунок 4.2 – Блок-схема підпрограми

Для отримання електричних імпульсів серця використовується датчик ЕКГ AD8232, і програмне забезпечення взаємодіє з цим датчиком, щоб зчитати значення. В процесі були розроблені функції та методи, які забезпечують взаємодію з датчиком. В додатку А наведений вихідний код, який реалізує цей процес.

Нижче наведено невеликий приклад використання датчика AD8232 з виводом даних тільки у Serial Port:

```
void setup() {

  // initialize the serial communication:
  Serial.begin(9600).
  pinMode(34, INPUT). // Setup for leads off detection LO +
  pinMode(35, INPUT). // Setup for leads off detection LO -
}

void loop() {

  if((digitalRead(34) == 1)||(digitalRead(35) == 1)){
    Serial.println('!').
  }
  else{
    // send the value of analog input 0:
      Serial.println(analogRead(A0)).
  }

  delay(1).
}
```

У функції “setup()” встановлюється швидкість передачі даних через Serial комунікацію (9600 біт/с) і налаштовуються піни 34 та 35 мікроконтролера ESP32 DEVKITV1 для виявлення роз'єднання контактів датчика (leads off detection).

У головній функції “loop()” здійснюється постійне опитування стану пінів 34 та 35, що відповідають за leads off detection. Якщо хоча б один з цих пінів має значення 1 (високий рівень напруги), то виводиться символ '!' на Serial порт. Це може вказувати на роз'єднання електродів датчика або проблеми з підключенням.

У протилежному випадку, якщо значення на пінах 34 та 35 дорівнюють 0 (низький рівень напруги), то виконується зчитування значення з аналогового піна A0, яке відповідає за зчитування сигналу. Отримане значення аналогового сигналу з датчика передається через Serial порт.

Таким чином, цей код забезпечує постійне зчитування електричних імпульсів серця з датчика ЕКГ AD8232 і виведення їх на Serial порт

мікроконтролера. Завдяки цьому можна спостерігати за змінами в електричній активності серця та аналізувати отримані дані.

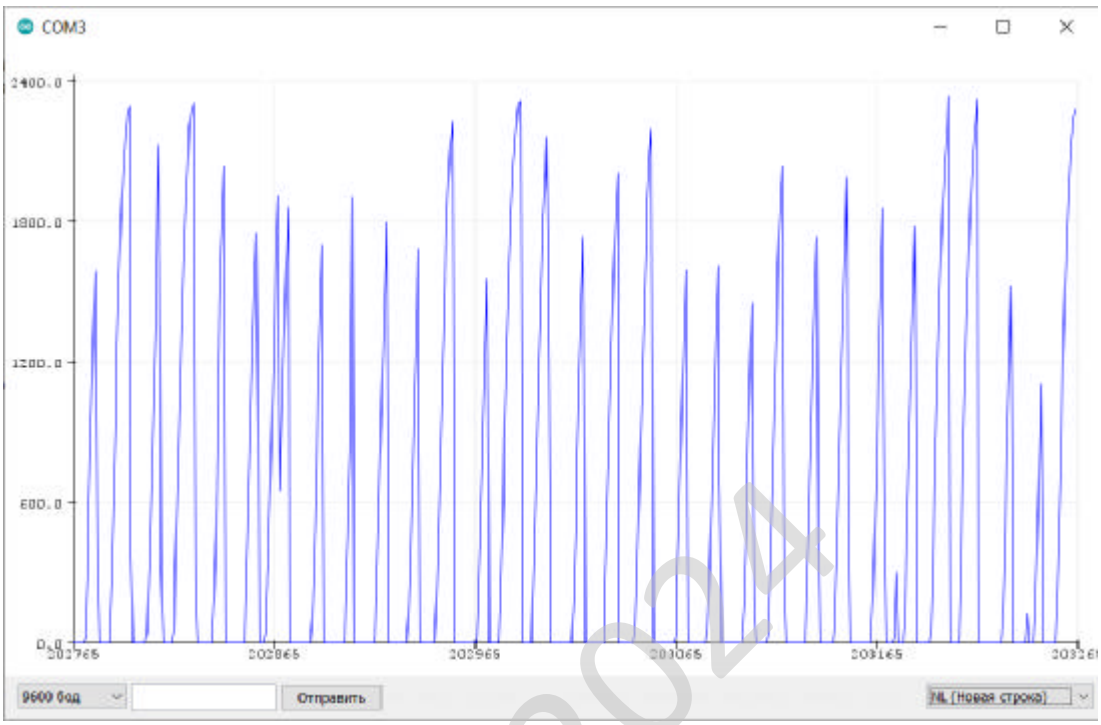

Рисунок 4.3 – Результат виконання прикладу

Далі буде проведено декілька дослідів, розміщуючи електроди в різні частини тіла для виявлення найкращого результату. Всі експерименти проводилися всередині приміщення, де система була розташована на столі. Піддослідний займав місце на стільці поруч з системою. Для збору даних про електричні імпульси серця використовувався датчик AD8232, а результати вимірювань передавалися на спеціальний веб-сайт Ubidots у реальному часі. Проте, як і у будь-якого пристрою, датчик мав свої помилки та похибки, тому і ще раз хотілось би наголосити, що датчик AD8232 не є професійним обладнанням, і тому можуть бути неточності у вимірюваннях та побудові графіків.

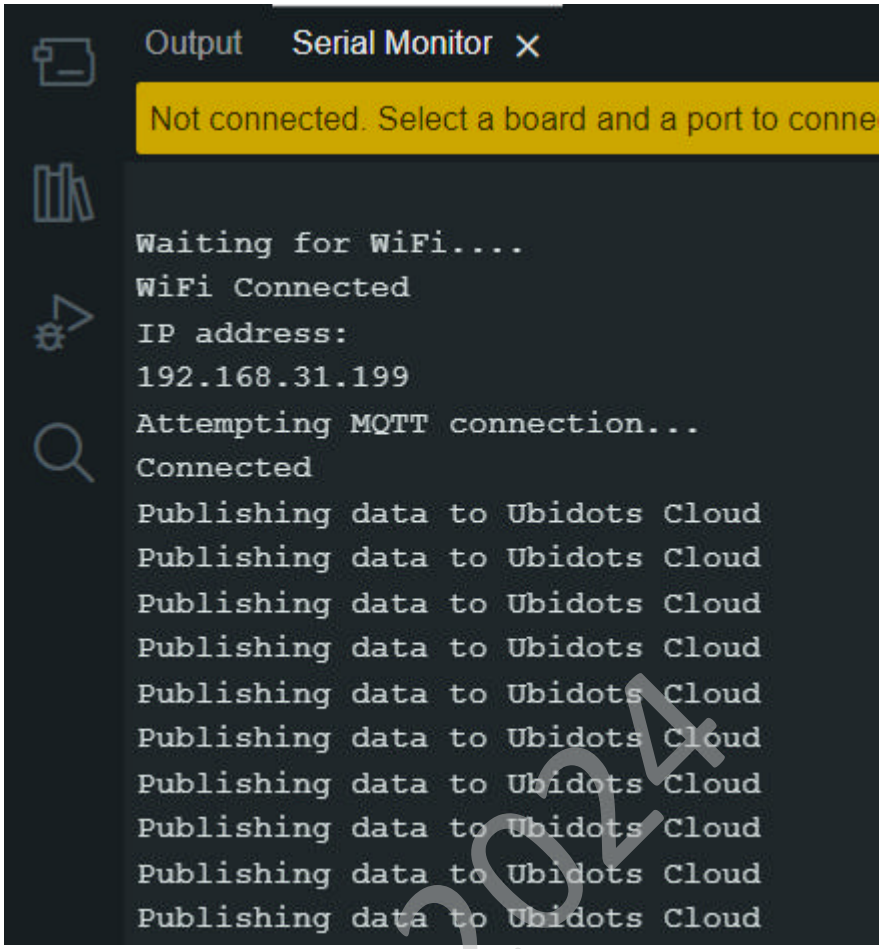

Рисунок 4.4 – Відображення того, що результати вимірювань передаються на веб-сайт

Дослід 1

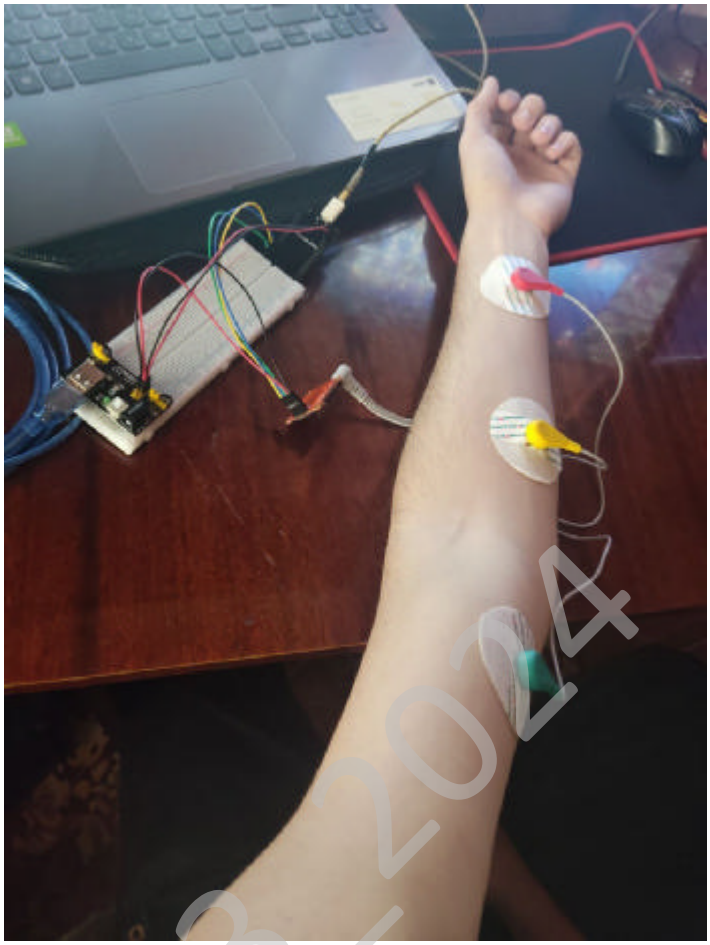

Рисунок 4.5 – Розташування електродів на лівій руці

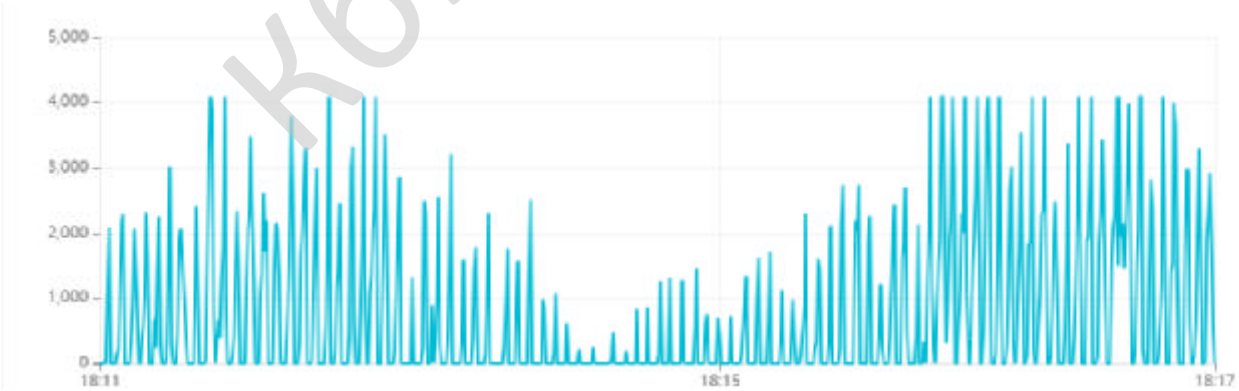

Рисунок 4.6 – Графік ЕКГ до першого досліду

Як видно з рисунка 4.6, результат доволі непоганий. Хоч всі електроди були розташовані на одній руці і були дуже близько один до одного, це не дуже

вплинуло на графік.

Дослід 2

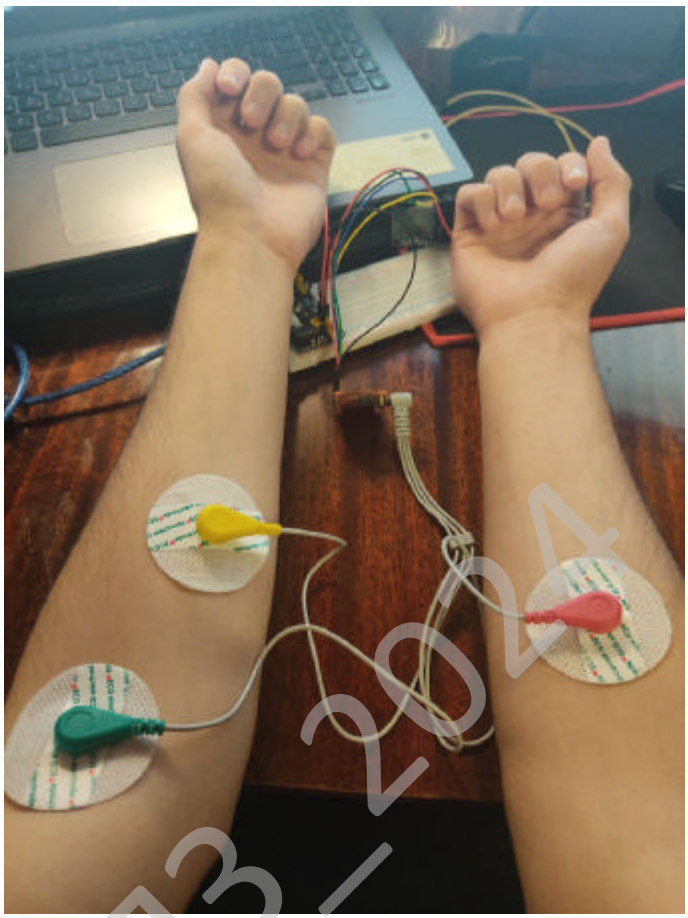

Рисунок 4.7 – Розташування електродів LA та RL на лівій руці, RA на правій руці

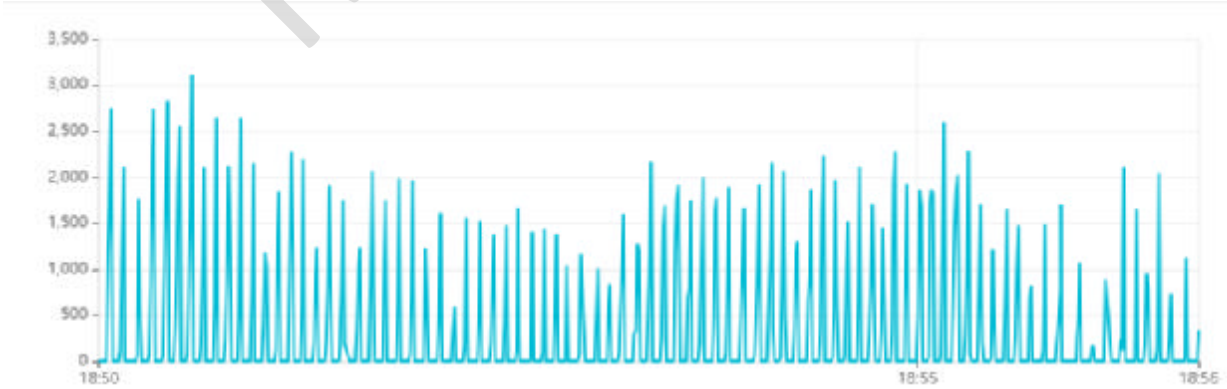

Рисунок 4.8 – Графік ЕКГ до другого досліду

Як видно з рисунка 4.8, графік майже рівномірний. Порівнюючи з

першим дослідом, в середині графіку немає провалу. Переміщення електрода RA на праву руку призвело до більш рівномірного графіка.

Дослід 3

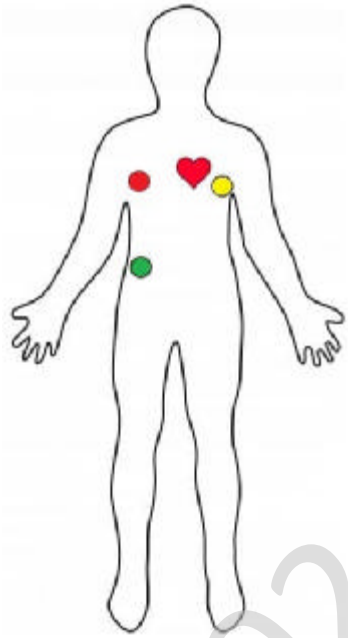

Рисунок 4.9 – Розташування електрода LA під серцем, RA над правою грудиною та RL на правому боці[10]

Рисунок 4.10 – Графік ЕКГ до третього досліду

Як видно з рисунка 4.10, початок та кінець графіку виглядають не дуже гарно, але середина тепер має високі значення, що правда, між ними майже немає малих значень.

Повторення Досліду 3

Рисунок 4.11 – Графік ЕКГ до повторного третього досліду

Як видно з рисунка 4.11, результат дуже схожий на правду, в місці з 11:14:30 і по 11:15:00 виглядає досить схожим, а вже далі пішов спад значень і в самому кінці знову високе значення. Якщо повторювати 3 дослід ще декілька разів, то можливо, буде результат схожий до нормального. Але, так як, це не професійний прилад, то такий графік ще дуже непогано виглядає.

4.2 Захист розробленого програмного забезпечення

Згідно закону України від 23.12.93 р. № 3792-XII “Про авторське право і суміжні права” в редакції від 16.07.01 року визначено терміни “комп’ютерна програма”. Відповідно до законодавства, комп'ютерні програми є об'єктом авторського права, тобто інтелектуальної власності. Основними елементами захисту програмного забезпечення є гарантування конфіденційності, цілісності та доступності даних і функціонування пристрою.

Однак варто зазначити, що використовуються загальнодоступні бібліотеки, які в процесі розробки системи використовувалися для забезпечення зв'язків з компонентами та модулями.

Основні бібліотеки:

– WiFi.h: вона дозволяє Arduino з'єднуватися з бездротовими мережами

Wi-Fi, використовуючи стандартні функції для підключення до точок доступу Wi-Fi і виконання різних мережевих операцій.

– PubSubClient.h: використовується для створення клієнта MQTT на платформах Arduino. MQTT є простим у використанні та ефективним протоколом для передачі повідомлень між пристроями у мережі IoT.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для того, щоб впровадити систему в промислову експлуатацію, потрібно мати чіткий алгоритм дій. Для своєї системи я склав алгоритм з 10 пунктів, які допоможуть ефективно впровадити систему ЕКГ моніторингу в промислову експлуатацію, забезпечивши її правильне функціонування та надійність. Даний алгоритм представлений нижче:

– Планування та підготовка.

– Вибір місця встановлення.

– Встановлення обладнання.

– Налаштування ПЗ.

– Тестування та налагодження.

– Інтеграція в промислову систему.

– Навчання персоналу та підтримка.

– Моніторинг та підтримка в експлуатації.

– Оновлення та розвиток.

– Оцінка ефективності та відгуків користувачів.

Кожний з цих пунктів складається із своїх підпунктів. Кожний пункт детально розписаний і показує, як саме потрібно ефективно впровадити систему в промислову експлуатацію. Дані підпункти наведені нижче:

1. Планування та підготовка:

– Визначення вимог і специфікацій: детально проаналізувати потреби промислового середовища. Ретельно визначити, які саме дані ЕКГ необхідні для моніторингу, як часто вони повинні збиратися, які вимоги до точності та надійності, і які можливості зберігання та обробки даних потрібні.

– Вибір обладнання та технологій: провести дослідження ринку, щоб визначити оптимальне обладнання та технології для виконання завдань.

– Оцінка бюджету та ресурсів: визначити бюджет, який потрібно виділити на розробку та впровадження системи ЕКГ моніторингу. Оцінити також необхідні людські та матеріальні ресурси для виконання проєкту. Створення плану проєкту: Розробити детальний план робіт, включаючи всі етапи розробки, випробувань, впровадження та підтримки системи.

2. Вибір місця встановлення:

– Оцінка доступності та безпеки: провести огляд промислового об'єкту для визначення найкращого місця для розміщення системи. Врахувати доступність живлення та можливість підключення до мережі зв'язку.

– Зручність розташування: важливо обрати місце, яке забезпечить оптимальний доступ до системи для користувачів, які моніторяться.

– Розгорнутість мережі зв'язку: перевірити доступність мережі зв'язку (наприклад, Wi-Fi або Bluetooth) в обраному місці.

3. Встановлення обладнання:

– Монтаж системи: ретельно встановити систему на відповідному місці, яке було визначено на попередньому етапі. Забезпечити надійне кріплення, щоб уникнути зміщень або пошкоджень в процесі експлуатації.

– Тестування працездатності: після встановлення та підключення провести тестування системи для перевірки працездатності.

– Кріплення: після успішного тестування закріпити обладнання на його постійному місці.

– Документація та дотримання стандартів: завершити цей етап шляхом оформлення документації про встановлення, включаючи схеми підключення, інструкції щодо обслуговування та безпеки.

4. Налаштування ПЗ:

– Розробка ПЗ: розробити ПЗ для збору, обробки та відображення даних з датчика ЕКГ.

– Налаштування передачі даних: налаштувати ПЗ для передачі даних з датчика ЕКГ через відповідний канал зв'язку.

– Налаштування параметрів: налаштувати ПЗ відповідно до специфічних потреб промислового середовища.

– Документація та навчання персоналу: забезпечити документацію щодо використання ПЗ та навчання персоналу, який буде працювати з системою.

5. Тестування та налагодження:

– Впевнитись, що система працює правильно та відповідає всім вимогам технічного завдання.

– Виявлення та усунення неполадок: якщо під час тестування виявлені будь-які неполадки або проблеми, то вжити заходи для їх усунення.

– Підтвердження відповідності вимогам: перевірити, що система відповідає всім вимогам, встановленим у технічному завданні.

– Документування результатів: записати всі результати тестування, виявлені неполадки та заходи їх усунення у спеціальній документації.

6. Інтеграція в промислову систему:

– Розробка інтерфейсів: розробити необхідні інтерфейси для забезпечення взаємодії системи з існуючим обладнанням або ПЗ.

– Тестування інтеграції: провести тестування взаємодії системи з існуючими промисловими системами.

– Підтвердження сумісності: переконатись, що система працює коректно в контексті промислового середовища.

– Документація та навчання персоналу: після успішної інтеграції системи з існуючим обладнанням надати документацію щодо використання та обслуговування інтегрованої системи.

7. Навчання персоналу та підтримка:

– Проведення навчальних курсів: організувати навчальні курси для персоналу, який буде відповідальним за роботу з системою.

– Надання технічної підтримки: забезпечити механізми для отримання технічної підтримки та консультацій з питань експлуатації системи.

– Моніторинг та підтримка в експлуатації: проводити постійний

моніторинг роботи системи та вчасно реагувати на будь-які виявлені проблеми.

– Звітність та аналіз даних: збирати дані про роботу системи та взаємодію з нею персоналу для подальшого аналізу та вдосконалення. Створювати звіти про продуктивність, ефективність та якість обслуговування, щоб ідентифікувати можливості для покращення.

8. Моніторинг та підтримка в експлуатації:

– Виявлення потенційних проблем: слідкувати за різноманітними параметрами роботи системи. Вчасно виявляти будь-які відхилення від норми, що можуть свідчити про потенційні проблеми.

– Вжиття заходів з усунення проблем: якщо під час моніторингу виявлені будь-які аномалії або проблеми, вжити відповідних заходів для їх усунення.

– Планове обслуговування: регулярно проводити обслуговування системи згідно з встановленим графіком.

– Оновлення та підтримка ПЗ: регулярно оновлювати ПЗ системи, враховуючи вимоги безпеки, функціональності та сумісності.

9. Оновлення та розвиток:

– Моніторинг технологічних тенденцій: проводити постійний моніторинг новітніх технологій та тенденцій у галузі промисловості. Слідкувати за інноваціями в області моніторингу здоров'я та технологій збору та аналізу даних.

– Впровадження нововведень: враховуючи зібрану інформацію про новітні розробки, потрібно розглянути можливості впровадження нововведень у систему.

– Тестування нововведень: перед впровадженням нововведень провести їх тестування на відповідність вимогам та стандартам.

10. Оцінка ефективності та відгуків користувачів:

– Збір даних про використання: проводити систематичний аналіз даних щодо використання системи. Оцінити частоту та тривалість використання,

типові сценарії застосування, а також інші параметри ефективності.

– Збір відгуків користувачів: отримувати фідбек від користувачів щодо їхнього досвіду використання системи.

– Впровадження покращень: на основі отриманих даних та відгуків користувачів визначити пріоритети для впровадження покращень у систему.

Розглянувши десять ключових пунктів інтеграції системи моніторингу ЕКГ в промислову експлуатацію, можна зробити важливі висновки щодо успішності та ефективності цього процесу. Впровадження системи вимагає комплексного підходу, який охоплює технічні, організаційні та людські аспекти.

Першим кроком у процесі інтеграції є вивчення вимог та аналіз потреб користувачів, що дозволяє визначити оптимальні технічні рішення. Важливою частиною процесу є вибір відповідних компонентів та їх інтеграція в промислове середовище. Під час інтеграції системи необхідно звернути увагу на аспекти безпеки, надійності та сумісності з існуючим обладнанням та ПЗ. Важливою частиною процесу є також навчання персоналу та підтримка після впровадження. Моніторинг ефективності та зворотній зв'язок від користувачів дозволяють постійно вдосконалювати систему та відповідати змінним потребам та вимогам. Впровадження системи в промислове середовище є складним процесом, але з правильним підходом та уважним врахуванням усіх аспектів може бути успішним та вигідним для підприємства.

Розроблено додаток з інформацією про авторські права для відображення коротких довідок про програму. Програма відображає інформацію про університет розробника, кафедру, предмет, спеціалізацію та іншу інформацію.

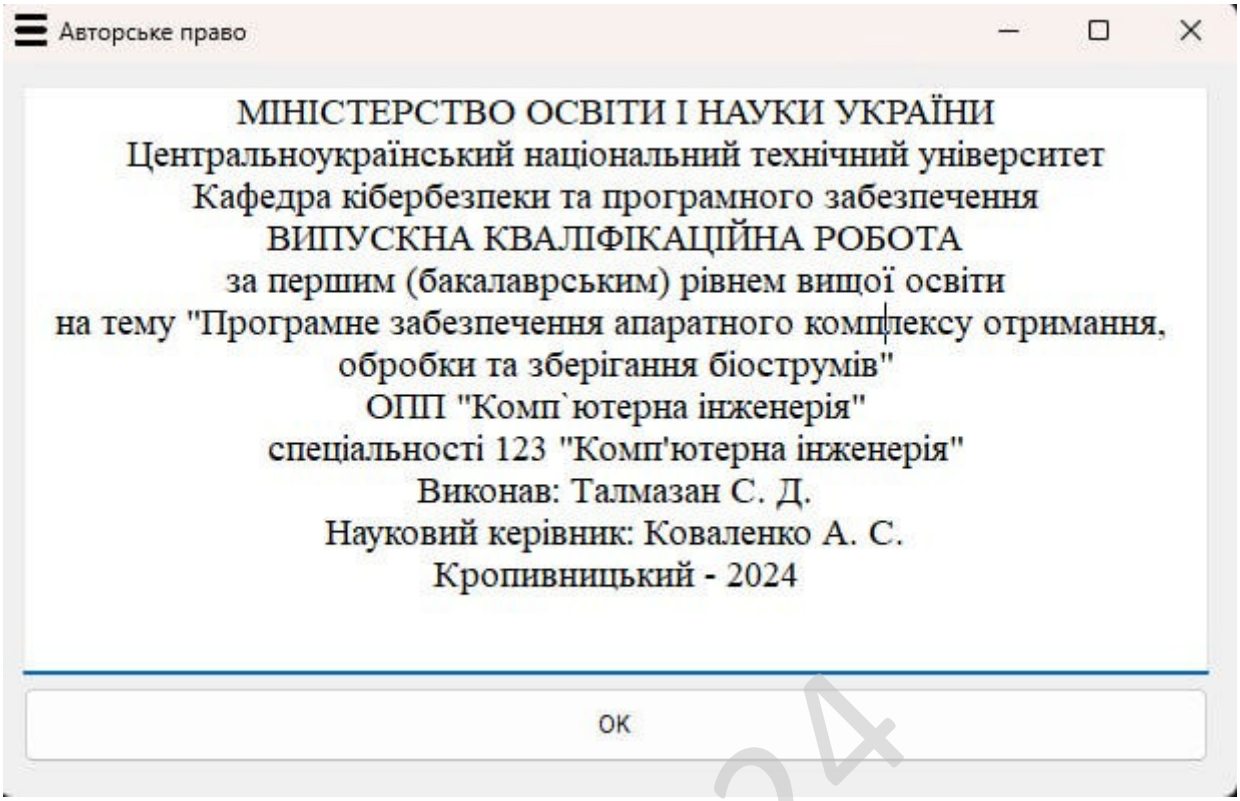

Рисунок 5.1 – Вікно розробника ПЗ

6 ОСНОВНІ ВИСНОВКИ

У даній роботі було проведено детальний аналіз стану сучасних систем, технологій, архітектур та програмних рішень в галузі біострумів і систем ЕКГ моніторингу. Це дозволило встановити основні недоліки і переваги існуючих систем і вибрати найбільш оптимальні засоби для розробки нової системи. У результаті обґрунтування вибору засобів для побудови системи та мови програмування була вибрана платформа ESP32 DEVKITV1 та датчик ЕКГ AD8232. Ці засоби мають високу надійність та широкі можливості для зчитування та обробки електричних сигналів серця. У процесі роботи було проведено дослідження основних принципів роботи мікроконтролера ESP32 DEVKITV1 та датчика ЕКГ AD8232, а також здійснена програмна реалізація системи зчитування даних з датчика та їх передачі на спеціальний веб-сайт моніторингу. Було перевірено функціональність системи та здійснено її налагодження.

Далі було проведено розробку структурної схеми системи, що включала в себе мікроконтролер, датчик ЕКГ, а також необхідні елементи для передачі даних. Також була розроблена функціональна схема, яка визначала основні функції системи, такі як зчитування, обробка та передача даних. Для перевірки правильності проектних і програмних рішень була проведена реалізація системи. Були розроблені блок-схеми та алгоритми функціонування системи, що дозволило підтвердити їх ефективність та вірність.

Крім того, робота включала розділ, присвячений реалізації системи та експериментальним даним, що підтверджують вірність проектних і програмних рішень. Цей етап дозволив перевірити працездатність системи та якість розробленого ПЗ. Отримані результати мають велике практичне значення. Розроблене ПЗ може бути застосоване в різних галузях, зокрема в медицині та біотехнологіях. У медицині воно може сприяти поліпшенню діагностики та

лікування різних хвороб, оскільки біоструми відіграють важливу роль в багатьох біологічних процесах. В біології та біотехнологіях воно може покращити якість досліджень, пов'язаних з вивченням біострумів.

Окремий розділ роботи присвячений впровадженню системи в промислову експлуатацію. В ньому розглядаються питання, пов'язані із впровадженням системи в реальні умови, підготовкою персоналу, організацією процесу впровадження та можливими перешкодами, які можуть виникнути під час цього процесу. Розроблене ПЗ готове до використання і може бути масштабоване для використання в різних установах та організаціях.

Загалом, дана робота є значним кроком у напрямку розвитку систем ЕКГ моніторингу та дослідження біострумів. Результати дослідження можуть бути використані як основа для подальших досліджень та вдосконалення системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ChatGPT [Електронний ресурс]. – Режим доступу: https://chat.openai.com

2. Ubidots [Електронний ресурс]. – Режим доступу: https://stem.ubidots.com

3. MB102 Breadboard Power Supply Module [Електронний ресурс]. – Режим доступу: https://www.circuits-diy.com/mb102-breadboard-power-supply-module

4. AD8232 (Rev. D) [Електронний ресурс]. – Режим доступу: https://www.analog.com/media/en/technical-documentation/data-sheets/ad8232.pdf

5. Модуль AD8232 для ЕКГ [Електронний ресурс]. – Режим доступу: https://arduino.ua/ru/prod4139-modyl-ad8232-dlya-ekg

6. Wi-Fi модуль DevKit V1 з ESP-32 [Електронний ресурс]. – Режим доступу: https://arduino.ua/prod3990-wi-fi-modyl-devkit-v1-s-esp-32

7. ESP32 WROOM DevKit v1 [Електронний ресурс]. – Режим доступу: http://wiki.amperka.ru/products:esp32-wroom-wifi-devkit-v1

8. ESP32_datasheet [Електронний ресурс]. – Режим доступу: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

9. The block diagram of ESP32 [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/figure/The-block-diagram-of-ESP32_fig3_353212187

10. IoT Based ECG Monitoring with AD8232 ECG Sensor & ESP32 [Електронний ресурс]. – Режим доступу: https://how2electronics.com/iot-ecg-monitoring-ad8232-sensor-esp32

11. ECG Monitoring system using AD8232 with ESP32 IOT Based [Електронний ресурс]. – Режим доступу: https://www.circuitschools.com/ecg-monitoring-system-using-ad8232-with-arduino-or-esp32-iot-based

12. Простой кардиомонитор своими руками [Електронний ресурс]. –

Режим доступу: https://cxem.net/medic/medic38.php

13. Real-Time Remote Patient Monitoring [Електронний ресурс]. – Режим доступу: https://www.mdpi.com/2076-3417/14/5/1876

14. Системи моніторингу «DiaCard» [Електронний ресурс]. – Режим доступу: https://solvaig.com/holter%D1%96vska-systema-diacard

15. CARDIOVIT AT-180 [Електронний ресурс]. – Режим доступу: https://www.schiller.ch/en/products/cardiovit-at-180-p34

16. Shenzhen Comen Medical Instruments Co., Ltd. [Електронний ресурс]. – Режим доступу: https://en.comen.com/pages/H12.html

17. EKG Machines [Електронний ресурс]. – Режим доступу: https://infiniummedical.com/ekg-machines

18. Portable ECG Machine heart monitor [Електронний ресурс]. – Режим доступу: https://www.visionflex.com/product/peripherals/electrocardiogram-devices/portable-ecg-heart-monitor

19. Lepu-Creative PC-80B [Електронний ресурс]. – Режим доступу: https://www.creative-sz.com/products/lepu-pc-80b-portable-ecg-monitor-easy-ekg-machine-handheld

20. Schiller AG [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Schiller_AG

21. Функціональна схема [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/

22. Arduino [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Arduino

23. Про авторське право і суміжні права [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2811-20#Text

24. Tom Jenkyns, Ben Stephenson. Fundamentals of Discrete Math for Computer Science. Springer. 2018. 514 с.

25. John Paul Mueller, Luca Massaron. Algorithms for Dummies. John Wiley & Sons, Inc. 2022. 451 с.

26. Marcello La Rocca. Advanced Algorithms and Data Structures. Manning Publications Co. 2021. 769 с.

27. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. The MIT Press. 2022 1677 с.

28. Кормен Томас Г. Вступ до алгоритмів: Переклад з англійської третього видання : [укр.] = Introduction to algorithms : Third Edition : [пер. з англ.] / Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест, Кліфорд Стайн, — К: К.І.С., 2019. — 1288 с.

29. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с. Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/9799

30. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

31. Смірнов О.А., Кавун С.В., Доренський О.П., Вялкова В.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 151 с.

32. Смірнов О.А., Стасєв Ю.В. Бараннік В.В. Захист інформації в автоматизованих системах управління. Навчальний посібник – Харків: ХУПС, 2015. – 264 с.

33. Смірнов О.А., Кузнецов О.О., Євсеєв С.П., Мелешко Є.В., Король О.Г. Методи та алгоритми симетричної криптографії. Навчальний посібник для студентів вищих навчальних закладів напрямів підготовки 8.050102 «Комп'ютерна інженерія». За ред. О.О. Кузнецова. Гриф "Навчальний посібник" надано у відповідності з листом Міністерства освіти і науки, молоді та спорту України від 26.04.2012 року № 1/11-5762. – Кіровоград: КНТУ 2012.

– 315 с.

34. Смірнов О.А., Мелешко Є.В., Семенов С.Г. Методи та засоби обробки сигналів і даних в інформаційних системах. Навчальний посібник для студентів вищих навчальних закладів напрямів підготовки 8.050102 «Комп'ютерна інженерія». За ред. О.А. Смірнова Гриф "Навчальний посібник" надано у відповідності з листом Міністерства освіти і науки, молоді та спорту України від 17.04.2012 року № 1/11-5249. – Кіровоград: КНТУ 2012. – 250 с.

35. Andrew S. Tanenbaum, Herbert Bos: Modern Operating Systems (Fourth Edition). March 10, 2014. Pearson Education, Inc. USA. May 2013. ISBN : 978-0-13-359162-0. 1136 pages.

36. Brian W. Kernigan, Rob Pike: UNIX Programming Enviornment. 1984. Bell Laboratories; PRENTICE-HALL; Whitehall Books. ISBN : 0-13-937699-2. 376 pages.

37. William Stallings: Operating Systems: Internals and Design Principles, 7th Editions. 2012. ISBN : 978-0-13-230998-1. 820 pages.

38. Harvey M. Deitel, Paul J. Deitel, David R. Choffnes: Operating Systems (3rd Edition)2003. ISBN : 978-0131828278. 1100 pages.

39. Alfred V. Aho, Compilers: Principles, Techniques, and Tools 2nd Edition, Kindle Edition / Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman/ - Pearson Education Limited, 2013 – 966 p.

40. Смірнов С.А./ Проектування комп'ютерних систем та мереж./ Смірнов С.А., Смірнов О.А., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. / Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

41. Смірнов С.А./ Розробка методу передтестової компіляції й розподілу доступу./ Смірнов С.А., Смірнов О.А., Коваленко О.В., Коваленко А.С. / Збірник наукових праць III міжнародної науково-практичної конференції "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

42. Мартін Р. Чиста архітектура. – Фабула, 2019. – 368 с

43. Yashavant Kanetkar. Data Structures Through C++. BPB Publications. 2022. 346 с.

44. Jonathan Sande. Data Structures & Algorithms in Dart. Razeware LLC. 2022. 425 с.

45. Боксолл Дж. Arduino Workshop, 2nd Edition: A Hands-on Introduction with 65 Projects. — San Francisco: No Starch Press, 2019. — 392 с.

46. Блюм Дж. Exploring Arduino: Tools and Techniques for Engineering Wizardry. — Indianapolis: Wiley, 2013. — 352 с.

47. Банзі М., Шіло М. Make: Getting Started with Arduino. — Sebastopol: Maker Media, Inc., 2014. — 262 с.

48. Смірнов С.А./ Розробка методу передтестової компіляції й розподілу доступу./ Смірнов С.А., Смірнов О.А., Коваленко О.В., Коваленко А.С. / Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

49. Johnson M. Hart/ Windows System Programming/ Johnson M. Hart / Addison-Wesley Professional; 4 edition, 2015 – 656 p.

50. Craft, Brock. Arduino Projects For Dummies. — Hoboken, NJ: For Dummies, 2013. — 408 p.

51. Система моніторингу «DIACARD» 2.1 (LE) [Електронний ресурс]. – Режим доступу: https://solvaig.com/holter-diacard-21-le

52. PC-80B [Електронний ресурс]. – Режим доступу: https://www.joom.com/uk/products/62f0ad89568345018604514 4?variant_id=62f0ad895683459f86045146

53. ESP32 DEVKITV1 [Електронний ресурс]. – Режим доступу: https://id.aliexpress.com/item/32806450471.html

54. ESP32 Pinout Reference [Електронний ресурс]. – Режим доступу:

https://lastminuteengineers.com/esp32-pinout-reference

55. ESP32 functional block diagram [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/figure/ESP32-functional-block-diagram_fig5_341446512

56. ECG Module AD8232 [Електронний ресурс]. – Режим доступу: https://www.amazon.co.uk/Youmile-Module-Measurement-Monitoring-Arduino/dp/B08216YR9H

57. Електроди датчика AD8232 [Електронний ресурс]. – Режим доступу: https://www.mdpi.com/2076-3417/14/5/1876

58. MB102 Breadboard Power Supply module [Електронний ресурс]. – Режим доступу: https://microcontrollerslab.com/mb102-breadboard-power-supply-module-pinout-and-how-to-use-it

59. Макетна плата та модуль живлення [Електронний ресурс]. – Режим доступу: https://id.aliexpress.com/item/32980640422.html?gatewayAdapt=glo2idn

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....4
5.5 Вимоги до надійності.....4
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....5
5.8.1 Обладнання.....5
5.8.2 Мова програмування.....5
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....6
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи апаратного комплексу отримання, обробки та зберігання біострумів.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №131-02 від 01.04.2024 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи апаратного комплексу отримання, обробки та зберігання біострумів.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– Підключення необхідних бібліотек для датчика ЕКГ та мікроконтролера.

– Налаштування пінів підключення для кожного компонента.

– Ініціалізація зв'язку з датчиком ЕКГ та мікроконтролером.

– Запуск зчитування значень електричних імпульсів серця з датчика ЕКГ.

– Перевірка успішності зчитування даних.

– Збереження значень електричних імпульсів серця для подальшого використання.

– Передача по Wi-Fi даних.

– Оновлення сторінки веб-сайту для відображення останніх результатів.

– Перевірка та аналіз отриманих даних з датчика ЕКГ.

– Завершення поточного циклу.

– Вимкнення модуля живлення та повторення алгоритму для продовження моніторингу.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel Core i3/12 Gb DDR4 /240 Gb/ GeForce MX230 2GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мові програмування Arduino.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 66 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 23.05.2024 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 04.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ А.С. Коваленко

Програмне забезпечення апаратного комплексу отримання, обробки та зберігання біострумів

Лістинг програми

Код документу 12

Загальна кількість аркушів: 21

Літера: РП

Кропивницький – 2024 року

Лістинг програми ECG_Monitoring.ino:

```
// Основна мета цього коду - надсилання даних з датчика
// до хмарної платформи Ubidots за допомогою протоколу MQTT

#include <WiFi.h>          // Бібліотека для роботи з WiFi
#include <PubSubClient.h>  // Бібліотека для роботи з MQTT

#define WIFISSID "KbPZ Free Wi-Fi"                    // SSID (ім'я Wi-Fi мережі)
KbPZ Free Wi-Fi | Xiaomi24G
#define PASSWORD ""                                   // Пароль для Wi-Fi
#define TOKEN "BBUS-zpmXkQFTZYGG3TTA5BKDgoDqiayHDW"  // Токен для з'єднання з
платформою Ubidots
#define MQTT_CLIENT_NAME "seregaclient"               // Ім'я MQTT клієнта

// Визначення міток (labels) для змінної та пристрою на платформі Ubidots
#define VARIABLE_LABEL "sensor"
#define DEVICE_LABEL "esp32"

#define SENSOR A0  // Визначення піна A0 як датчика

char mqttBroker[] = "industrial.api.ubidots.com";  // Адреса MQTT брокера
char payload[100];                                 // Буфер для зберігання
повідомлення
char topic[150];                                   // Тема
char str_sensor[10];                               // Місце для зберігання
значень для надсилання

// Створення клієнта Wi-Fi та клієнта MQTT для з'єднання з платформою Ubidots
WiFiClient ubidots;
PubSubClient client(ubidots);

// Функція-обробник повідомлень MQTT, яка виводить отримане повідомлення та тему
на Serial порт
void callback(char* topic, byte* payload, unsigned int length) {

  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;

  Serial.write(payload, length);
  Serial.println(topic);
}

// reconnect() відновлює з'єднання з MQTT-брокером. Вона перевіряє, чи вдалося
// підключитися до брокера, якщо ні, то вона повторює спробу підключення через 2
секунди
void reconnect() {

  // Повторюємо, доки не з'єднаємось
  while (!client.connected()) {

    Serial.println("Attempting MQTT connection...");

    // Спроба підключення
    if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {

      Serial.println("Connected");

    } else {

      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");

      delay(2000);
    }
```

```
  }
}

void setup() {

  Serial.begin(115200);  // Порт ініціалізується з швидкістю 115200

  WiFi.begin(WIFISSID, PASSWORD);  // Підключення до Wi-Fi мережі з
використанням вказаних SSID і пароля

  pinMode(SENSOR, INPUT);  // Пін SENSOR встановлюється в режим введення (INPUT)

  // Чекаємо, доки не буде встановлене з'єднання з Wi-Fi, і виводиться IP-адреса
  Serial.println();
  Serial.print("Waiting for WiFi...");

  while (WiFi.status() != WL_CONNECTED) {

    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Встановлюються сервер MQTT (брокер) і функція зворотного виклику (callback)
для обробки повідомлень
  client.setServer(mqttBroker, 1883);
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }

  // Створюється тема (topic) та повідомлення (payload) для надсилання даних до
платформи Ubidots
  sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
  sprintf(payload, "%s", "");
  sprintf(payload, "{\"%s\":", VARIABLE_LABEL);

  float sensor = analogRead(SENSOR);  // Зчитування значення з датчика

  // 4 - мінімальна ширина, 2 - точність; значення з плаваючою комою копіюється
в str_sensor
  dtostrf(sensor, 4, 2, str_sensor);  // Конвертація значення в рядковий формат

  sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor);  // Додавання
значення до повідомлення
  Serial.println("Publishing data to Ubidots Cloud");

  client.publish(topic, payload);  // Публікація повідомлення на платформу
Ubidots
  client.loop();                   // Обробка вхідних та вихідних повідомлень
MQTT

  delay(500);
}
```

Лістинг програми mqtt_large_message.ino:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = ".......";
const char* password = ".......";
const char* mqtt_server = "broker.mqtt-dashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Find out how many bottles we should generate lyrics for
  String topicStr(topic);
  int bottleCount = 0; // assume no bottles unless we correctly parse a value
from the topic
  if (topicStr.indexOf('/') >= 0) {
    // The topic includes a '/', we'll try to read the number of bottles from
just after that
    topicStr.remove(0, topicStr.indexOf('/')+1);
    // Now see if there's a number of bottles after the '/'
    bottleCount = topicStr.toInt();
  }

  if (bottleCount > 0) {
    // Work out how big our resulting message will be
    int msgLen = 0;
    for (int i = bottleCount; i > 0; i--) {
      String numBottles(i);
      msgLen += 2*numBottles.length();
      if (i == 1) {
        msgLen += 2*String(" green bottle, standing on the wall\n").length();
      } else {
```

```
        msgLen += 2*String(" green bottles, standing on the wall\n").length();
      }
      msgLen += String("And if one green bottle should accidentally
fall\nThere'll be ").length();
      switch (i) {
      case 1:
        msgLen += String("no green bottles, standing on the wall\n\n").length();
        break;
      case 2:
        msgLen += String("1 green bottle, standing on the wall\n\n").length();
        break;
      default:
        numBottles = i-1;
        msgLen += numBottles.length();
        msgLen += String(" green bottles, standing on the wall\n\n").length();
        break;
      };
    }

    // Now we can start to publish the message
    client.beginPublish("greenBottles/lyrics", msgLen, false);
    for (int i = bottleCount; i > 0; i--) {
      for (int j = 0; j < 2; j++) {
        client.print(i);
        if (i == 1) {
          client.print(" green bottle, standing on the wall\n");
        } else {
          client.print(" green bottles, standing on the wall\n");
        }
      }
      client.print("And if one green bottle should accidentally fall\nThere'll
be ");
      switch (i) {
      case 1:
        client.print("no green bottles, standing on the wall\n\n");
        break;
      case 2:
        client.print("1 green bottle, standing on the wall\n\n");
        break;
      default:
        client.print(i-1);
        client.print(" green bottles, standing on the wall\n\n");
        break;
      };
    }
    // Now we're done!
    client.endPublish();
  }
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
      client.subscribe("greenBottles/#");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
```

```
      delay(5000);
    }
  }
}

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin as an
output
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Лістинг програми mqtt_client.ino:

```
#include "PubSubClient.h"
#include "Arduino.h"

PubSubClient::PubSubClient() {
    this->_state = MQTT_DISCONNECTED;
    this->_client = NULL;
    this->stream = NULL;
    setCallback(NULL);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}

PubSubClient::PubSubClient(Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}

PubSubClient::PubSubClient(IPAddress addr, uint16_t port, Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(addr, port);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(IPAddress addr, uint16_t port, Client& client,
Stream& stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(addr,port);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
```

```
}
PubSubClient::PubSubClient(IPAddress addr, uint16_t port,
MQTT_CALLBACK_SIGNATURE, Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(addr, port);
    setCallback(callback);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(IPAddress addr, uint16_t port,
MQTT_CALLBACK_SIGNATURE, Client& client, Stream& stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(addr,port);
    setCallback(callback);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}

PubSubClient::PubSubClient(uint8_t *ip, uint16_t port, Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(ip, port);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(uint8_t *ip, uint16_t port, Client& client, Stream&
stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(ip,port);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(uint8_t *ip, uint16_t port, MQTT_CALLBACK_SIGNATURE,
Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(ip, port);
    setCallback(callback);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(uint8_t *ip, uint16_t port, MQTT_CALLBACK_SIGNATURE,
Client& client, Stream& stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(ip,port);
    setCallback(callback);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
```

```
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}

PubSubClient::PubSubClient(const char* domain, uint16_t port, Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(domain,port);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(const char* domain, uint16_t port, Client& client,
Stream& stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(domain,port);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(const char* domain, uint16_t port,
MQTT_CALLBACK_SIGNATURE, Client& client) {
    this->_state = MQTT_DISCONNECTED;
    setServer(domain,port);
    setCallback(callback);
    setClient(client);
    this->stream = NULL;
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}
PubSubClient::PubSubClient(const char* domain, uint16_t port,
MQTT_CALLBACK_SIGNATURE, Client& client, Stream& stream) {
    this->_state = MQTT_DISCONNECTED;
    setServer(domain,port);
    setCallback(callback);
    setClient(client);
    setStream(stream);
    this->bufferSize = 0;
    setBufferSize(MQTT_MAX_PACKET_SIZE);
    setKeepAlive(MQTT_KEEPALIVE);
    setSocketTimeout(MQTT_SOCKET_TIMEOUT);
}

PubSubClient::~PubSubClient() {
  free(this->buffer);
}

boolean PubSubClient::connect(const char *id) {
    return connect(id,NULL,NULL,0,0,0,0,1);
}

boolean PubSubClient::connect(const char *id, const char *user, const char
*pass) {
    return connect(id,user,pass,0,0,0,0,1);
}

boolean PubSubClient::connect(const char *id, const char* willTopic, uint8_t
willQos, boolean willRetain, const char* willMessage) {
    return connect(id,NULL,NULL,willTopic,willQos,willRetain,willMessage,1);
}
```

```
boolean PubSubClient::connect(const char *id, const char *user, const char
*pass, const char* willTopic, uint8_t willQos, boolean willRetain, const char*
willMessage) {
    return connect(id,user,pass,willTopic,willQos,willRetain,willMessage,1);
}

boolean PubSubClient::connect(const char *id, const char *user, const char
*pass, const char* willTopic, uint8_t willQos, boolean willRetain, const char*
willMessage, boolean cleanSession) {
    if (!connected()) {
        int result = 0;


        if(_client->connected()) {
            result = 1;
        } else {
            if (domain != NULL) {
                result = _client->connect(this->domain, this->port);
            } else {
                result = _client->connect(this->ip, this->port);
            }
        }

        if (result == 1) {
            nextMsgId = 1;
            // Leave room in the buffer for header and variable length field
            uint16_t length = MQTT_MAX_HEADER_SIZE;
            unsigned int j;

#if MQTT_VERSION == MQTT_VERSION_3_1
            uint8_t d[9] = {0x00,0x06,'M','Q','I','s','d','p', MQTT_VERSION};
#define MQTT_HEADER_VERSION_LENGTH 9
#elif MQTT_VERSION == MQTT_VERSION_3_1_1
            uint8_t d[7] = {0x00,0x04,'M','Q','T','T',MQTT_VERSION};
#define MQTT_HEADER_VERSION_LENGTH 7
#endif
            for (j = 0;j<MQTT_HEADER_VERSION_LENGTH;j++) {
                this->buffer[length++] = d[j];
            }

            uint8_t v;
            if (willTopic) {
                v = 0x04|(willQos<<3)|(willRetain<<5);
            } else {
                v = 0x00;
            }
            if (cleanSession) {
                v = v|0x02;
            }

            if(user != NULL) {
                v = v|0x80;

                if(pass != NULL) {
                    v = v|(0x80>>1);
                }
            }
            this->buffer[length++] = v;

            this->buffer[length++] = ((this->keepAlive) >> 8);
            this->buffer[length++] = ((this->keepAlive) & 0xFF);

            CHECK_STRING_LENGTH(length,id)
            length = writeString(id,this->buffer,length);
            if (willTopic) {
                CHECK_STRING_LENGTH(length,willTopic)
                length = writeString(willTopic,this->buffer,length);
                CHECK_STRING_LENGTH(length,willMessage)
                length = writeString(willMessage,this->buffer,length);
```

```
            }

            if(user != NULL) {
                CHECK_STRING_LENGTH(length,user)
                length = writeString(user,this->buffer,length);
                if(pass != NULL) {
                    CHECK_STRING_LENGTH(length,pass)
                    length = writeString(pass,this->buffer,length);
                }
            }

            write(MQTTCONNECT,this->buffer,length-MQTT_MAX_HEADER_SIZE);

            lastInActivity = lastOutActivity = millis();

            while (!_client->available()) {
                unsigned long t = millis();
                if (t-lastInActivity >= ((int32_t) this->socketTimeout*1000UL))
{
                    _state = MQTT_CONNECTION_TIMEOUT;
                    _client->stop();
                    return false;
                }
            }
            uint8_t llen;
            uint32_t len = readPacket(&llen);

            if (len == 4) {
                if (buffer[3] == 0) {
                    lastInActivity = millis();
                    pingOutstanding = false;
                    _state = MQTT_CONNECTED;
                    return true;
                } else {
                    _state = buffer[3];
                }
            }
            _client->stop();
        } else {
            _state = MQTT_CONNECT_FAILED;
        }
        return false;
    }
    return true;
}

// reads a byte into result
boolean PubSubClient::readByte(uint8_t * result) {
   uint32_t previousMillis = millis();
   while(!_client->available()) {
     yield();
     uint32_t currentMillis = millis();
     if(currentMillis - previousMillis >= ((int32_t) this->socketTimeout *
1000)){
       return false;
     }
   }
   *result = _client->read();
   return true;
}

// reads a byte into result[*index] and increments index
boolean PubSubClient::readByte(uint8_t * result, uint16_t * index){
  uint16_t current_index = *index;
  uint8_t * write_address = &(result[current_index]);
  if(readByte(write_address)){
    *index = current_index + 1;
    return true;
  }
```

```
  return false;
}

uint32_t PubSubClient::readPacket(uint8_t* lengthLength) {
    uint16_t len = 0;
    if(!readByte(this->buffer, &len)) return 0;
    bool isPublish = (this->buffer[0]&0xF0) == MQTTPUBLISH;
    uint32_t multiplier = 1;
    uint32_t length = 0;
    uint8_t digit = 0;
    uint16_t skip = 0;
    uint32_t start = 0;

    do {
        if (len == 5) {
            // Invalid remaining length encoding - kill the connection
            _state = MQTT_DISCONNECTED;
            _client->stop();
            return 0;
        }
        if(!readByte(&digit)) return 0;
        this->buffer[len++] = digit;
        length += (digit & 127) * multiplier;
        multiplier <<=7; //multiplier *= 128
    } while ((digit & 128) != 0);
    *lengthLength = len-1;

    if (isPublish) {
        // Read in topic length to calculate bytes to skip over for Stream
writing
        if(!readByte(this->buffer, &len)) return 0;
        if(!readByte(this->buffer, &len)) return 0;
        skip = (this->buffer[*lengthLength+1]<<8)+this->buffer[*lengthLength+2];
        start = 2;
        if (this->buffer[0]&MQTTQOS1) {
            // skip message id
            skip += 2;
        }
    }
    uint32_t idx = len;

    for (uint32_t i = start;i<length;i++) {
        if(!readByte(&digit)) return 0;
        if (this->stream) {
            if (isPublish && idx-*lengthLength-2>skip) {
                this->stream->write(digit);
            }
        }

        if (len < this->bufferSize) {
            this->buffer[len] = digit;
            len++;
        }
        idx++;
    }

    if (!this->stream && idx > this->bufferSize) {
        len = 0; // This will cause the packet to be ignored.
    }
    return len;
}

boolean PubSubClient::loop() {
    if (connected()) {
        unsigned long t = millis();
        if ((t - lastInActivity > this->keepAlive*1000UL) || (t -
lastOutActivity > this->keepAlive*1000UL)) {
            if (pingOutstanding) {
                this->_state = MQTT_CONNECTION_TIMEOUT;
```

```
                _client->stop();
                return false;
            } else {
                this->buffer[0] = MQTTPINGREQ;
                this->buffer[1] = 0;
                _client->write(this->buffer,2);
                lastOutActivity = t;
                lastInActivity = t;
                pingOutstanding = true;
            }
        }
        if (_client->available()) {
            uint8_t llen;
            uint16_t len = readPacket(&llen);
            uint16_t msgId = 0;
            uint8_t *payload;
            if (len > 0) {
                lastInActivity = t;
                uint8_t type = this->buffer[0]&0xF0;
                if (type == MQTTPUBLISH) {
                    if (callback) {
                        uint16_t tl = (this->buffer[llen+1]<<8)+this-
>buffer[llen+2]; /* topic length in bytes */
                        memmove(this->buffer+llen+2,this->buffer+llen+3,tl); /*
move topic inside buffer 1 byte to front */
                        this->buffer[llen+2+tl] = 0; /* end the topic as a 'C'
string with \x00 */
                        char *topic = (char*) this->buffer+llen+2;
                        // msgId only present for QOS>0
                        if ((this->buffer[0]&0x06) == MQTTQOS1) {
                            msgId = (this->buffer[llen+3+tl]<<8)+this-
>buffer[llen+3+tl+1];
                            payload = this->buffer+llen+3+tl+2;
                            callback(topic,payload,len-llen-3-tl-2);

                            this->buffer[0] = MQTTPUBACK;
                            this->buffer[1] = 2;
                            this->buffer[2] = (msgId >> 8);
                            this->buffer[3] = (msgId & 0xFF);
                            _client->write(this->buffer,4);
                            lastOutActivity = t;

                        } else {
                            payload = this->buffer+llen+3+tl;
                            callback(topic,payload,len-llen-3-tl);
                        }
                    }
                } else if (type == MQTTPINGREQ) {
                    this->buffer[0] = MQTTPINGRESP;
                    this->buffer[1] = 0;
                    _client->write(this->buffer,2);
                } else if (type == MQTTPINGRESP) {
                    pingOutstanding = false;
                }
            } else if (!connected()) {
                // readPacket has closed the connection
                return false;
            }
        }
        return true;
    }
    return false;
}

boolean PubSubClient::publish(const char* topic, const char* payload) {
    return publish(topic,(const uint8_t*)payload, payload ? strnlen(payload,
this->bufferSize) : 0,false);
}
```

```
boolean PubSubClient::publish(const char* topic, const char* payload, boolean
retained) {
    return publish(topic,(const uint8_t*)payload, payload ? strnlen(payload,
this->bufferSize) : 0,retained);
}

boolean PubSubClient::publish(const char* topic, const uint8_t* payload,
unsigned int plength) {
    return publish(topic, payload, plength, false);
}

boolean PubSubClient::publish(const char* topic, const uint8_t* payload,
unsigned int plength, boolean retained) {
    if (connected()) {
        if (this->bufferSize < MQTT_MAX_HEADER_SIZE + 2+strnlen(topic, this-
>bufferSize) + plength) {
            // Too long
            return false;
        }
        // Leave room in the buffer for header and variable length field
        uint16_t length = MQTT_MAX_HEADER_SIZE;
        length = writeString(topic,this->buffer,length);

        // Add payload
        uint16_t i;
        for (i=0;i<plength;i++) {
            this->buffer[length++] = payload[i];
        }

        // Write the header
        uint8_t header = MQTTPUBLISH;
        if (retained) {
            header |= 1;
        }
        return write(header,this->buffer,length-MQTT_MAX_HEADER_SIZE);
    }
    return false;
}

boolean PubSubClient::publish_P(const char* topic, const char* payload, boolean
retained) {
    return publish_P(topic, (const uint8_t*)payload, payload ? strnlen(payload,
this->bufferSize) : 0, retained);
}

boolean PubSubClient::publish_P(const char* topic, const uint8_t* payload,
unsigned int plength, boolean retained) {
    uint8_t llen = 0;
    uint8_t digit;
    unsigned int rc = 0;
    uint16_t tlen;
    unsigned int pos = 0;
    unsigned int i;
    uint8_t header;
    unsigned int len;
    int expectedLength;

    if (!connected()) {
        return false;
    }

    tlen = strnlen(topic, this->bufferSize);

    header = MQTTPUBLISH;
    if (retained) {
        header |= 1;
    }
    this->buffer[pos++] = header;
    len = plength + 2 + tlen;
```

```
    do {
        digit = len  & 127; //digit = len %128
        len >>= 7; //len = len / 128
        if (len > 0) {
            digit |= 0x80;
        }
        this->buffer[pos++] = digit;
        llen++;
    } while(len>0);

    pos = writeString(topic,this->buffer,pos);

    rc += _client->write(this->buffer,pos);

    for (i=0;i<plength;i++) {
        rc += _client->write((char)pgm_read_byte_near(payload + i));
    }

    lastOutActivity = millis();

    expectedLength = 1 + llen + 2 + tlen + plength;

    return (rc == expectedLength);
}

boolean PubSubClient::beginPublish(const char* topic, unsigned int plength,
boolean retained) {
    if (connected()) {
        // Send the header and variable length field
        uint16_t length = MQTT_MAX_HEADER_SIZE;
        length = writeString(topic,this->buffer,length);
        uint8_t header = MQTTPUBLISH;
        if (retained) {
            header |= 1;
        }
        size_t hlen = buildHeader(header, this->buffer, plength+length-
MQTT_MAX_HEADER_SIZE);
        uint16_t rc = _client->write(this->buffer+(MQTT_MAX_HEADER_SIZE-
hlen),length-(MQTT_MAX_HEADER_SIZE-hlen));
        lastOutActivity = millis();
        return (rc == (length-(MQTT_MAX_HEADER_SIZE-hlen)));
    }
    return false;
}

int PubSubClient::endPublish() {
 return 1;
}

size_t PubSubClient::write(uint8_t data) {
    lastOutActivity = millis();
    return _client->write(data);
}

size_t PubSubClient::write(const uint8_t *buffer, size_t size) {
    lastOutActivity = millis();
    return _client->write(buffer,size);
}

size_t PubSubClient::buildHeader(uint8_t header, uint8_t* buf, uint16_t length)
{
    uint8_t lenBuf[4];
    uint8_t llen = 0;
    uint8_t digit;
    uint8_t pos = 0;
    uint16_t len = length;
    do {

        digit = len  & 127; //digit = len %128
```

```
        len >>= 7; //len = len / 128
        if (len > 0) {
            digit |= 0x80;
        }
        lenBuf[pos++] = digit;
        llen++;
    } while(len>0);

    buf[4-llen] = header;
    for (int i=0;i<llen;i++) {
        buf[MQTT_MAX_HEADER_SIZE-llen+i] = lenBuf[i];
    }
    return llen+1; // Full header size is variable length bit plus the 1-byte
fixed header
}

boolean PubSubClient::write(uint8_t header, uint8_t* buf, uint16_t length) {
    uint16_t rc;
    uint8_t hlen = buildHeader(header, buf, length);

#ifdef MQTT_MAX_TRANSFER_SIZE
    uint8_t* writeBuf = buf+(MQTT_MAX_HEADER_SIZE-hlen);
    uint16_t bytesRemaining = length+hlen;  //Match the length type
    uint8_t bytesToWrite;
    boolean result = true;
    while((bytesRemaining > 0) && result) {
        bytesToWrite = (bytesRemaining >
MQTT_MAX_TRANSFER_SIZE)?MQTT_MAX_TRANSFER_SIZE:bytesRemaining;
        rc = _client->write(writeBuf,bytesToWrite);
        result = (rc == bytesToWrite);
        bytesRemaining -= rc;
        writeBuf += rc;
    }
    return result;
#else
    rc = _client->write(buf+(MQTT_MAX_HEADER_SIZE-hlen),length+hlen);
    lastOutActivity = millis();
    return (rc == hlen+length);
#endif
}

boolean PubSubClient::subscribe(const char* topic) {
    return subscribe(topic, 0);
}

boolean PubSubClient::subscribe(const char* topic, uint8_t qos) {
    size_t topicLength = strnlen(topic, this->bufferSize);
    if (topic == 0) {
        return false;
    }
    if (qos > 1) {
        return false;
    }
    if (this->bufferSize < 9 + topicLength) {
        // Too long
        return false;
    }
    if (connected()) {
        // Leave room in the buffer for header and variable length field
        uint16_t length = MQTT_MAX_HEADER_SIZE;
        nextMsgId++;
        if (nextMsgId == 0) {
            nextMsgId = 1;
        }
        this->buffer[length++] = (nextMsgId >> 8);
        this->buffer[length++] = (nextMsgId & 0xFF);
        length = writeString((char*)topic, this->buffer,length);
        this->buffer[length++] = qos;
```

```
        return write(MQTTSUBSCRIBE|MQTTQOS1,this->buffer,length-
MQTT_MAX_HEADER_SIZE);
    }
    return false;
}

boolean PubSubClient::unsubscribe(const char* topic) {
      size_t topicLength = strnlen(topic, this->bufferSize);
    if (topic == 0) {
        return false;
    }
    if (this->bufferSize < 9 + topicLength) {
        // Too long
        return false;
    }
    if (connected()) {
        uint16_t length = MQTT_MAX_HEADER_SIZE;
        nextMsgId++;
        if (nextMsgId == 0) {
            nextMsgId = 1;
        }
        this->buffer[length++] = (nextMsgId >> 8);
        this->buffer[length++] = (nextMsgId & 0xFF);
        length = writeString(topic, this->buffer,length);
        return write(MQTTUNSUBSCRIBE|MQTTQOS1,this->buffer,length-
MQTT_MAX_HEADER_SIZE);
    }
    return false;
}

void PubSubClient::disconnect() {
    this->buffer[0] = MQTTDISCONNECT;
    this->buffer[1] = 0;
    _client->write(this->buffer,2);
    _state = MQTT_DISCONNECTED;
    _client->flush();
    _client->stop();
    lastInActivity = lastOutActivity = millis();
}

uint16_t PubSubClient::writeString(const char* string, uint8_t* buf, uint16_t
pos) {
    const char* idp = string;
    uint16_t i = 0;
    pos += 2;
    while (*idp) {
        buf[pos++] = *idp++;
        i++;
    }
    buf[pos-i-2] = (i >> 8);
    buf[pos-i-1] = (i & 0xFF);
    return pos;
}


boolean PubSubClient::connected() {
    boolean rc;
    if (_client == NULL ) {
        rc = false;
    } else {
        rc = (int)_client->connected();
        if (!rc) {
            if (this->_state == MQTT_CONNECTED) {
                this->_state = MQTT_CONNECTION_LOST;
                _client->flush();
                _client->stop();
            }
        } else {
            return this->_state == MQTT_CONNECTED;
```

```
        }
    }
    return rc;
}

PubSubClient& PubSubClient::setServer(uint8_t * ip, uint16_t port) {
    IPAddress addr(ip[0],ip[1],ip[2],ip[3]);
    return setServer(addr,port);
}

PubSubClient& PubSubClient::setServer(IPAddress ip, uint16_t port) {
    this->ip = ip;
    this->port = port;
    this->domain = NULL;
    return *this;
}

PubSubClient& PubSubClient::setServer(const char * domain, uint16_t port) {
    this->domain = domain;
    this->port = port;
    return *this;
}

PubSubClient& PubSubClient::setCallback(MQTT_CALLBACK_SIGNATURE) {
    this->callback = callback;
    return *this;
}

PubSubClient& PubSubClient::setClient(Client& client){
    this->_client = &client;
    return *this;
}

PubSubClient& PubSubClient::setStream(Stream& stream){
    this->stream = &stream;
    return *this;
}

int PubSubClient::state() {
    return this->_state;
}

boolean PubSubClient::setBufferSize(uint16_t size) {
    if (size == 0) {
        // Cannot set it back to 0
        return false;
    }
    if (this->bufferSize == 0) {
        this->buffer = (uint8_t*)malloc(size);
    } else {
        uint8_t* newBuffer = (uint8_t*)realloc(this->buffer, size);
        if (newBuffer != NULL) {
            this->buffer = newBuffer;
        } else {
            return false;
        }
    }
    this->bufferSize = size;
    return (this->buffer != NULL);
}

uint16_t PubSubClient::getBufferSize() {
    return this->bufferSize;
}
PubSubClient& PubSubClient::setKeepAlive(uint16_t keepAlive) {
    this->keepAlive = keepAlive;
    return *this;
}
PubSubClient& PubSubClient::setSocketTimeout(uint16_t timeout) {
```

```
    this->socketTimeout = timeout;
    return *this;
}
```

Лістинг програми WiFi_client.ino:

```
#include "utility/wifi_drv.h"
#include "WiFi.h"

extern "C" {
  #include "utility/wl_definitions.h"
  #include "utility/wl_types.h"
  #include "utility/debug.h"
}

// XXX: don't make assumptions about the value of MAX_SOCK_NUM.
int16_t     WiFiClass::_state[MAX_SOCK_NUM] = { NA_STATE, NA_STATE, NA_STATE,
NA_STATE };
uint16_t    WiFiClass::_server_port[MAX_SOCK_NUM] = { 0, 0, 0, 0 };

WiFiClass::WiFiClass()
{
}

void WiFiClass::init()
{
    WiFiDrv::wifiDriverInit();
}

uint8_t WiFiClass::getSocket()
{
    for (uint8_t i = 0; i < MAX_SOCK_NUM; ++i)
    {
        if (WiFiClass::_server_port[i] == 0)
        {
             return i;
        }
    }
    return NO_SOCKET_AVAIL;
}

char* WiFiClass::firmwareVersion()
{
      return WiFiDrv::getFwVersion();
}

int WiFiClass::begin(char* ssid)
{
      uint8_t status = WL_IDLE_STATUS;
      uint8_t attempts = WL_MAX_ATTEMPT_CONNECTION;

   if (WiFiDrv::wifiSetNetwork(ssid, strlen(ssid)) != WL_FAILURE)
   {
         do
         {
               delay(WL_DELAY_START_CONNECTION);
               status = WiFiDrv::getConnectionStatus();
         }
         while ((( status == WL_IDLE_STATUS)||(status == WL_SCAN_COMPLETED))&&(-
-attempts>0));
   }else
   {
         status = WL_CONNECT_FAILED;
   }
   return status;
}

int WiFiClass::begin(char* ssid, uint8_t key_idx, const char *key)
```

```
{
      uint8_t status = WL_IDLE_STATUS;
      uint8_t attempts = WL_MAX_ATTEMPT_CONNECTION;

      // set encryption key
   if (WiFiDrv::wifiSetKey(ssid, strlen(ssid), key_idx, key, strlen(key)) !=
WL_FAILURE)
   {
         do
         {
               delay(WL_DELAY_START_CONNECTION);
               status = WiFiDrv::getConnectionStatus();
         }while ((( status == WL_IDLE_STATUS)||(status ==
WL_SCAN_COMPLETED))&&(--attempts>0));
   }else{
         status = WL_CONNECT_FAILED;
   }
   return status;
}

int WiFiClass::begin(char* ssid, const char *passphrase)
{
      uint8_t status = WL_IDLE_STATUS;
      uint8_t attempts = WL_MAX_ATTEMPT_CONNECTION;

    // set passphrase
    if (WiFiDrv::wifiSetPassphrase(ssid, strlen(ssid), passphrase,
strlen(passphrase))!= WL_FAILURE)
    {
         do
         {
               delay(WL_DELAY_START_CONNECTION);
               status = WiFiDrv::getConnectionStatus();
         }
         while ((( status == WL_IDLE_STATUS)||(status == WL_SCAN_COMPLETED))&&(-
-attempts>0));
    }else{
      status = WL_CONNECT_FAILED;
    }
    return status;
}

void WiFiClass::config(IPAddress local_ip)
{
      WiFiDrv::config(1, (uint32_t)local_ip, 0, 0);
}

void WiFiClass::config(IPAddress local_ip, IPAddress dns_server)
{
      WiFiDrv::config(1, (uint32_t)local_ip, 0, 0);
      WiFiDrv::setDNS(1, (uint32_t)dns_server, 0);
}

void WiFiClass::config(IPAddress local_ip, IPAddress dns_server, IPAddress
gateway)
{
      WiFiDrv::config(2, (uint32_t)local_ip, (uint32_t)gateway, 0);
      WiFiDrv::setDNS(1, (uint32_t)dns_server, 0);
}

void WiFiClass::config(IPAddress local_ip, IPAddress dns_server, IPAddress
gateway, IPAddress subnet)
{
      WiFiDrv::config(3, (uint32_t)local_ip, (uint32_t)gateway,
(uint32_t)subnet);
      WiFiDrv::setDNS(1, (uint32_t)dns_server, 0);
}

void WiFiClass::setDNS(IPAddress dns_server1)
```

```
{
      WiFiDrv::setDNS(1, (uint32_t)dns_server1, 0);
}

void WiFiClass::setDNS(IPAddress dns_server1, IPAddress dns_server2)
{
      WiFiDrv::setDNS(2, (uint32_t)dns_server1, (uint32_t)dns_server2);
}

int WiFiClass::disconnect()
{
    return WiFiDrv::disconnect();
}

uint8_t* WiFiClass::macAddress(uint8_t* mac)
{
      uint8_t* _mac = WiFiDrv::getMacAddress();
      memcpy(mac, _mac, WL_MAC_ADDR_LENGTH);
    return mac;
}

IPAddress WiFiClass::localIP()
{
      IPAddress ret;
      WiFiDrv::getIpAddress(ret);
      return ret;
}

IPAddress WiFiClass::subnetMask()
{
      IPAddress ret;
      WiFiDrv::getSubnetMask(ret);
      return ret;
}

IPAddress WiFiClass::gatewayIP()
{
      IPAddress ret;
      WiFiDrv::getGatewayIP(ret);
      return ret;
}

char* WiFiClass::SSID()
{
    return WiFiDrv::getCurrentSSID();
}

uint8_t* WiFiClass::BSSID(uint8_t* bssid)
{
      uint8_t* _bssid = WiFiDrv::getCurrentBSSID();
      memcpy(bssid, _bssid, WL_MAC_ADDR_LENGTH);
    return bssid;
}

int32_t WiFiClass::RSSI()
{
    return WiFiDrv::getCurrentRSSI();
}

uint8_t WiFiClass::encryptionType()
{
    return WiFiDrv::getCurrentEncryptionType();
}


int8_t WiFiClass::scanNetworks()
{
      uint8_t attempts = 10;
      uint8_t numOfNetworks = 0;
```

```
      if (WiFiDrv::startScanNetworks() == WL_FAILURE)
            return WL_FAILURE;
      do
      {
            delay(2000);
            numOfNetworks = WiFiDrv::getScanNetworks();
      }
      while (( numOfNetworks == 0)&&(--attempts>0));
      return numOfNetworks;
}

char* WiFiClass::SSID(uint8_t networkItem)
{
      return WiFiDrv::getSSIDNetoworks(networkItem);
}

int32_t WiFiClass::RSSI(uint8_t networkItem)
{
      return WiFiDrv::getRSSINetoworks(networkItem);
}

uint8_t WiFiClass::encryptionType(uint8_t networkItem)
{
    return WiFiDrv::getEncTypeNetowrks(networkItem);
}

uint8_t WiFiClass::status()
{
    return WiFiDrv::getConnectionStatus();
}

int WiFiClass::hostByName(const char* aHostname, IPAddress& aResult)
{
      return WiFiDrv::getHostByName(aHostname, aResult);
}

WiFiClass WiFi;
```