МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК* (ім'я, ПРІЗВИЩЕ)
« 10 » 06 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Система моніторингу серцебиття з використанням платформи Arduino*

Група: *3-013* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти (підпис) *Максим СОЛТАНОВ* (ім'я, ПРІЗВИЩЕ)

Керівник роботи (підпис) *Владислав СОБЧУК* (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки (підпис) *Оксана ОСАДЧА* (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітній ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК* (ім'я, ПРІЗВИЩЕ)
« *01* » *05* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

Солтанову Максиму Євгенійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи *Система моніторингу серцебиття з використанням платформи Arduino*

Керівник роботи *Собчук Владислав Олегович*
викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з *01.03.2025* по *15.06.2025*

3. Вихідні дані до роботи *Мікроконтролер Arduino Leonardo. Програмне забезпечення Arduino Studio. Інструментальний підсилювач INA333.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *Біологія серця та екг у сучасному світі. Цифрова частина: мікроконтролер arduino. Проєктування аналогової частини. Інтерфейс та візуалізація даних.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Презентація Microsoft PowerPoint*

6. Консультанти розділів роботи (проекту)

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Узгодження технічного завдання*	*01.03.2025*	
2	*Огляд літератури по темі*	*15.03.2025*	
	кваліфікаційної роботи		
3	Біологія серця та екг у сучасному світі	*28.04.2025*	
4	Цифрова частина: мікроконтролер *arduino*	*14.05.2025*	
5	Проєктування аналогової частини	*22.05.2025*	
6	Інтерфейс та візуалізація даних	*26.05.2025*	
7	*Оформлення пояснювальної записки*	*06.06.2025*	
8	*Захист кваліфікаційної роботи*		

Здобувач освіти _____ *Максим СОЛТАНОВ*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Владислав СОБЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)

StrikePlagiarism.com

Дата звіту 6/9/2025
Дата редагування ---

Документ прийнятий

Звіт подібності

метадані

Назва організації
Ukrainian national aviation university
Заголовок
Солтанов М_3-013_2025_КПІ
Автор Науковий керівник / Експерт
СолтановГринченко О
підрозділ
Криворізький Фаховий коледж

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

25	9343	71846
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система моніторингу серцебиття з використанням платформи *Arduino*» викладена на 62 с., містить 20 рис., 1 табл. 16 використаних літературних джерел.

ARDUINO, МІКРОКОНТРОЛЕР, ДИСКРЕТИЗАЦІЯ, ШВИДКЕ ПЕРЕТВОРЕННЯ ФУР'Є, ВИМІРЮВАННЯ ПУЛЬСУ

Кваліфікаційна робота присвячена розробці пристрою, що реєструє змінну різницю потенціалів між двома точками при роботі серця, за отриманими даний, проводиться розрахунок пульсу пацієнта, а також відображаються отримані даний на моніторі ПК. Також детально досліджена платформа *Arduino*, її технічні можливості та особливості програмування вбудованого мікроконтролера.

Досліджено методи дискретизації аналогового сигналу в цифровий. Розглянуто алгоритм швидкого перетворення Фурье. Розроблено алгоритм вимірювання пульсу за отриманим сигналом.

Тема кваліфікаційної роботи є актуальною і має теоретичне і практичне значення.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

АЦП – Аналого-цифровий перетворювач

ВЗП – Вентильний затворний пристрій (якщо використовується) ЖКД – Рідкокристалічний дисплей

I/O – Введення/Виведення (*Input*/*Output*)

КМОН – Комплементарний метал-оксид-напівпровідник (технологія) МК – Мікроконтролер

ОЗП – Оперативний запам'ятовуючий пристрій

ПЗП – Постійний запам'ятовуючий пристрій

ПК – персональний комп‘ютер

ЕКГ – електро кардіограмма

USB – Universal Serial Bus

АЦП – аналого цифровий перетворювач

ОП – операційний підсилювач

ШПФ – швидке перетворювання Фур’є

ЕОМ – електронна обчислювальна машина

jSSC – Java Simple Serial Connector

LGPL – Lesser General Public License

ВСТУП

У сучасному світі ми постійно стикаємося з новими викликами та завданнями, і їх кількість лише зростає. Замість того, щоб чекати, що це колись закінчиться, нам необхідно адаптуватися, шукати інноваційні рішення або використовувати вже існуючі знання. Одним із ефективних інструментів для цього є мікроконтролер *Arduino*.

Застосування технологій у медицині та саморозвитку

Стрімкий розвиток медичних технологій дозволяє нам все глибше вивчати людський організм. Ідея проста: використовувати прилади для моніторингу різних біометричних показників, таких як біоструми, біопотенціали серця, мозку, м'язів та інших органів. Уявіть собі людину, буквально "вкриту" датчиками, що дозволяє отримати повну картину стану здоров'я.

Далі, на основі цих даних, можна застосовувати індивідуалізовані методи впливу: від медикаментозної терапії до фізичних вправ, дихальних технік, коригування режиму сну та харчування. Головна мета – відстежувати, наскільки ті чи інші дії є корисними для організму, і коригувати їх для досягнення оптимальних результатів.

Моніторинг та покращення здібностей

Ці технології використовуються не лише для лікування, але й для розвитку фізичних, розумових та інших здібностей. Наприклад, спортсмени відстежують біоструми м'язів, щоб визначити ефективність тренувань. Аналогічний принцип застосовується для тренування серцевого м'яза за допомогою електрокардіограми (ЕКГ).

Сьогодні набули широкої популярності так звані "нейротренажери" або "пристрої для покращення когнітивних функцій", які використовують різні світлові, звукові та тактильні стимули для покращення пам'яті, здатності до навчання, емоційного стану тощо. Для оцінки ефективності роботи таких пристроїв часто використовують електроенцефалограф (ЕЕГ), що реєструє електричну активність мозку.

Розширені застосування: від поліграфа до нейроінтерфейсів

Технології, засновані на моніторингу біопотенціалів, знаходять своє застосування і в інших сферах, таких як поліграф (детектор брехні), який реєструє фізіологічні зміни в організмі у відповідь на запитання.

Крім того, пристрої на основі ЕЕГ вже активно використовуються для керування зовнішніми технологіями (наприклад, комп'ютерами) "силою думки". Ця сфера відома як інтерфейси мозок-комп'ютер (*BCI*) і відкриває величезні перспективи для людей з обмеженими можливостями та для нових форм взаємодії з технологіями. Розробка персонального пристрою для моніторингу серця

Як приклад практичного застосування, я працюю над створенням пристрою для моніторингу власного серця. Цей пристрій складається з трьох ключових компонентів:

- Аналогова частина: електронна схема, що перетворює біологічні сигнали в електричні.
- Цифрова частина: мікроконтролер *Arduino*, який обробляє отримані дані.
- Програмна частина: програмне забезпечення для персонального комп'ютера, що візуалізує та аналізує дані.

Я детальніше розгляну ці складові далі.

РОЗДІЛ1
БІОЛОГІЯ СЕРЦЯ ТА ЕКГ У СУЧАСНОМУ СВІТІ

Однією з ключових переваг електрокардіограми (ЕКГ) для оцінки роботи серця є її швидкість та доступність. Сучасні ЕКГ-апарати миттєво відображають дані на екрані, а потім їх можна зберегти в цифровій формі або роздрукувати. Це дозволяє лікарям або навіть кваліфікованому медичному персоналу оперативно отримати результати і зробити первинні висновки щодо стану серця. Таке швидке реагування є критично важливим, особливо в невідкладних ситуаціях.

1.1 Основи розуміння електрокардіограми

На перший погляд, ЕКГ-крива може здатися надзвичайно складною: безперервні коливання, численні позначки та терміни. Може здатися, що лише досвідчений кардіолог здатен її розшифрувати. Однак це не зовсім так. Хоча аналіз ЕКГ дійсно вимагає уважності, концентрації та розуміння певних алгоритмів, базові принципи її інтерпретації можна опанувати.

Навички читання ЕКГ необхідні не лише кардіологам. Лікарі загальної практики, фельдшери та парамедики (особливо на швидкій допомозі) повинні вміти швидко аналізувати ЕКГ. Рання діагностика та інтерпретація ЕКГ, ще до прибуття пацієнта до лікарні, може врятувати життя, наприклад, при серцевому нападі, дозволяючи своєчасно розпочати невідкладну допомогу.

Сьогодні все більше людей прагнуть розуміти власне здоров'я, і це бажання часто поширюється на можливість самостійно розбиратися в ЕКГ. Хоча медичні довідники можуть здатися надто складними для неспеціалістів через велику кількість термінів, це не привід відмовлятися від ідеї розширити свої знання про кардіологію. Першим кроком є розуміння того, що саме відображає ЕКГ-лінія.

1.2 Що показує електрокардіограма

З фізіологічної точки зору, робота серця – це циклічний процес деполяризації

та реполяризації міокарда. Простіше кажучи, це постійна зміна фаз скорочення та розслаблення серцевого м'яза. Під час деполяризації клітини міокарда збуджуються і скорочуються, а під час реполяризації вони відновлюються і розслабляються. ЕКГ реєструє електричні сигнали, що супроводжують ці процеси.

Для розуміння патернів ЕКГ необхідно знати її основні елементи: - Зубець: піднесена або заглиблена частина кривої відносно ізолінії (горизонтальної осі).

- Сегмент: пряма лінія між двома сусідніми зубцями.

- Інтервал: поєднання зубців та сегментів.

Знання цих базових елементів дозволяє почати розшифровку складної, на перший погляд, ЕКГ-кривої.

1.3 Аналіз елементів ЕКГ: Сучасний підхід

При інтерпретації електрокардіограми (ЕКГ) амплітуда (сила) електричного сигналу відображається на вертикальній осі, а його тривалість – на горизонтальній. Кожна хвиля в серцевому циклі позначається латинською літерою і відповідає за проходження електричного імпульсу через певну частину серця, що дозволяє детально аналізувати його функцію.

Основні елементи ЕКГ та їх значення:

- Зубець *P*: Цей зубець відображає деполяризацію передсердь, тобто їхнє електричне збудження, яке передує скороченню.

- У нормі цей зубець має позитивне спрямування, округлу форму, висоту до 2,5 мм та тривалість до 0,1 секунди.

- Патологічні зміни можуть включати:

- Високий, загострений зубець *P*: часто вказує на гіпертрофію правого передсердя (збільшення його розміру).

- Подвійний (роздвоєний) зубець *P* може бути ознакою гіпертрофії лівого передсердя.

- Зубець *Q* характеризує початкову деполяризацію міжшлуночкової перегородки.
    - Зазвичай цей зубець невеликий і негативний (спрямований вниз), тривалістю лише близько 0.03 секунди.
    - У дітей глибокий зубець *Q* може бути нормою і не є приводом для занепокоєння.
- Зубець *R*: Це найбільший за амплітудою зубець, що описує поширення електричних сигналів по міокарду шлуночків і їхнє скорочення.
    - Його тривалість зазвичай не перевищує тривалості зубця *Q*.
- Зубець *S*: Показує завершення деполяризації шлуночків. Як і зубець *Q*, він негативний і зазвичай має меншу глибину – до 2 мм.
- Зубець *T* відповідає за реполяризацію міокарда, тобто процес відновлення електричного потенціалу після скорочення.
    - Зазвичай зубець *T* позитивний, гладкий і не піднімається вище ізолінії більш ніж на третину амплітуди зубця *R*. Його тривалість становить від 0.16 до 0.24 секунди.
    - Високий зубець *T* може свідчити про гіперкаліємію (надлишковий вміст калію в крові).
    - Інвертований або увігнутий зубець *T* із загостреною симетричною формою є характерною ознакою інфаркту міокарда та потребує термінової медичної допомоги.
- Зубець *U*: Рідко реєструється на ЕКГ і зазвичай його висота не перевищує 2 мм.
    - Може з'являтися у спортсменів після інтенсивного фізичного навантаження.
    - В інших випадках може свідчити про брадикардію (уповільнене серцебиття).

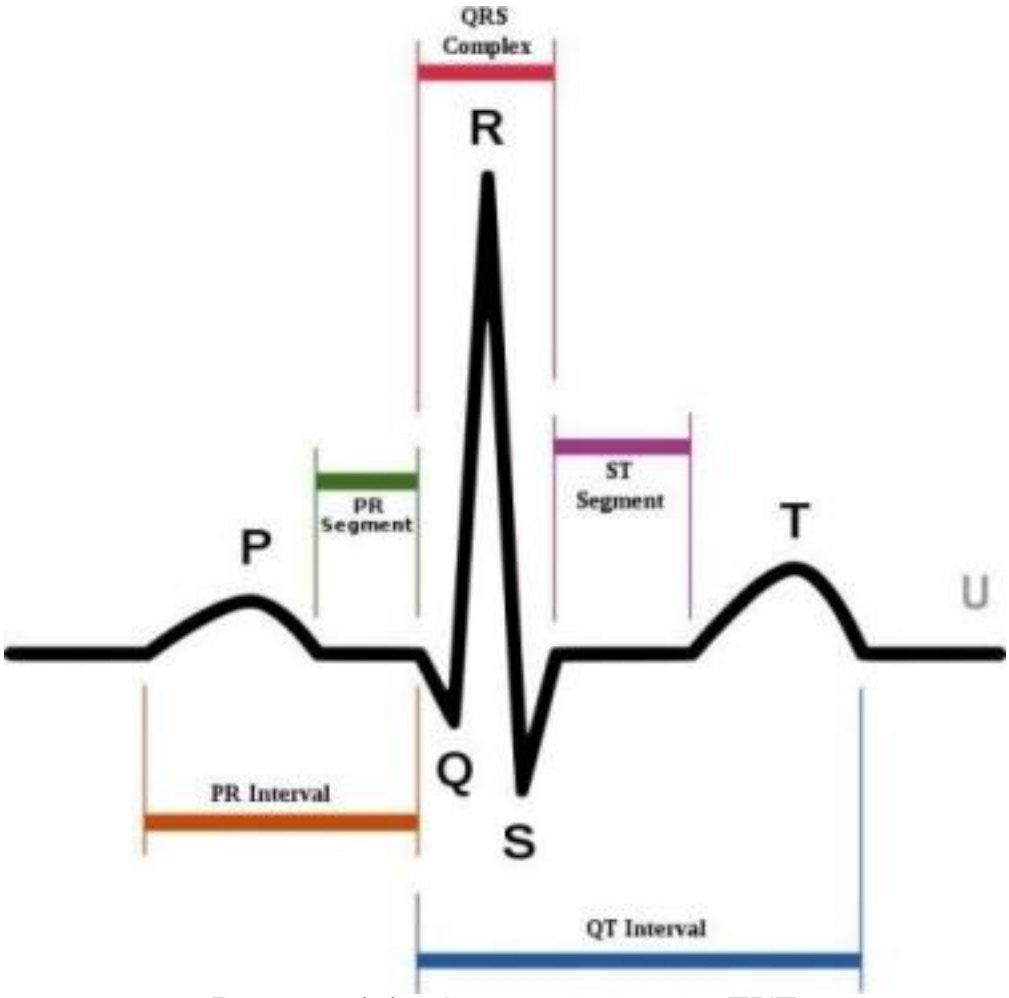

Рисунок 1.1 - Серцевий цикл на ЕКГ

Для наочного розуміння послідовності подій у серці під час реєстрації ЕКГ, розглянемо схематичне зображення, що ілюструє взаємозв'язок між електричною активністю та фазами скорочення та розслаблення серця.

Схема серцевого циклу на ЕКГ:

- *P*-хвиля: Початок електричної активності в передсердях.

- Інтервал *PQ*: Час проходження імпульсу від передсердь до шлуночків. - Комплекс *QRS*: Швидке поширення імпульсу по шлуночках (їх деполяризація та скорочення).

- Сегмент *ST*: Фаза плато (стабілізації) електричної активності шлуночків. - Зубець *T*: Реполяризація (відновлення) шлуночків.

- Зубець *U* (якщо присутній): Завершальна фаза реполяризації.

Аналіз сегментів ЕКГ

Оцінка функції серця також включає аналіз сегментів ЕКГ. Кожен сегмент вимірюється від кінця одного зубця до початку наступного. Найбільш важливими є сегменти *PQ* та *ST*.

- Сегмент *PQ* (або *PR*): Від кінця зубця *P* до початку комплексу *QRS*. Відображає час затримки імпульсу в атріовентрикулярному вузлі, дозволяючи передсердям завершити скорочення перед скороченням шлуночків.

- Сегмент *ST*: Від кінця комплексу *QRS* до початку зубця *T*. Цей сегмент відображає фазу повного збудження шлуночків, коли всі їхні клітини вже деполяризовані.

Аналіз цих сегментів включає оцінку їхньої довжини та відхилення від ізоелектричної лінії (горизонтальної осі). У нормі відхилення амплітуди цих сегментів не повинно перевищувати 1 мм. Будь-які значні відхилення або зміни тривалості можуть бути ознакою серйозних порушень, таких як ішемія міокарда (недостатнє кровопостачання) або аритмія (порушення ритму серця), оскільки тривалість сегментів безпосередньо пов'язана з серцевим ритмом.

1.4 Оцінка роботи міокарда за часовими інтервалами

Для точного аналізу ЕКГ критично важливо зосередитися на тривалості інтервалів, оскільки кожен з них відображає швидкість поширення електричного імпульсу у певній частині серця та реакцію міокарда. Наприклад, стандартна тривалість інтервалу *QT* складає приблизно 0.45 секунди. Його подовження може свідчити про серйозні стани, такі як ішемія (недостатнє кровопостачання) або атеросклероз (ущільнення артерій).

Тривалість інтервалів безпосередньо характеризує роботу міокарда в часі. Визначити частоту серцевих скорочень (ЧСС) або пульс за ЕКГ-графіком досить просто. Для цього використовується інтервал *RR* – відстань між двома послідовними найвищими зубцями *R*. У здорової дорослої людини в стані спокою цей показник

становить 70-80 ударів за хвилину. При цьому відстань між зубцями *R* не повинна

відрізнятися від середнього значення більш ніж на 10%. Такий ритм називається синусовим ритмом, що є нормою. Будь-які інші типи серцевого ритму можуть вказувати на патологічні зміни, які потребують подальшого обстеження та визначення джерела аномального збудження (кардіостимулятора).

1.5 Сучасний алгоритм інтерпретації ЕКГ

Пам'ятати всі деталі ЕКГ може бути складно, але існують стандартизовані підходи, які полегшують процес. Сучасна інтерпретація ЕКГ, що використовується фахівцями, ґрунтується на комплексній оцінці, яка включає такі ключові етапи:

- Оцінка серцевого ритму та провідності: Визначення регулярності серцевих скорочень, джерела імпульсу та швидкості його поширення.
- Визначення "електричної осі серця": Це узагальнений напрямок електричної активності серця, що допомагає оцінити його положення та можливі перевантаження певних відділів.
- Аналіз функції передсердь: Оцінка зубця *P* та інтервалу *PQ* для виявлення порушень у роботі передсердь та їхньому зв'язку зі шлуночками. - Характеристика елементів комплексу *QRS-T*: Детальний аналіз форми, амплітуди та тривалості зубців *Q*, *R*, *S* та *T*, а також сегментів *ST*, що відображає роботу шлуночків.
- Формулювання кардіографічного висновку: Узагальнення всіх отриманих даних для встановлення діагнозу або визначення подальших дій. Важливою передумовою для достовірної інтерпретації ЕКГ є перевірка правильності її запису. На початку дослідження подається контрольний сигнал – стандартна напруга 1 мілівольт, що повинна викликати відхилення ЕКГ-лінії на 10 мм. Якщо цей крок не виконаний або відхилення не відповідає нормі, запис ЕКГ вважається некоректним і не може бути використаний для діагностики. Правильна інтерпретація ЕКГ неможлива без врахування індивідуальних фізіологічних особливостей пацієнта, якізначно впливають на ЕКГ-патерни. До таких

особливостей належать вік, стать, статура, зріст та наявність хронічних

захворювань. Нехтування цими даними може призвести до помилкової інтерпретації відхилень як ознак серцевих захворювань.

Наприклад, "електрична вісь серця" може приблизно вказувати на анатомічне положення органу в грудній клітці, його розмір та форму. Проте у худих людей електрична вісь часто має вертикальне положення, тоді як у людей з надмірною вагою – горизонтальне. В обох випадках це вважається нормою. Крім того, глибока інтерпретація ЕКГ вимагає знання численних медичних термінів, що описують ознаки серцевих захворювань, таких як фібриляція передсердь, екстрасистолія, тріпотіння передсердь тощо, що є сферою компетенції кваліфікованих медичних фахівців.

РОЗДІЛ 2
ЦИФРОВА ЧАСТИНА: МІКРОКОНТРОЛЕР *ARDUINO*

У нашому проєкті цифрова частина пристрою реалізована на базі мікроконтролера *Arduino Leonardo* (рис. 2.1).

Рисунок 2.1 - Мікроконтролер *Arduino Mega*

Ось приклад одного з потужних мікроконтролерів *Arduino*, який часто використовується в складних проєктах.

Arduino – це не просто електронний конструктор, а ціла екосистема, що надає доступну та гнучку платформу для швидкої розробки електронних пристроїв. Вона однаково зручна як для початківців, так і для досвідчених інженерів. Популярність

Arduino зумовлена простотою мови програмування (що базується на *C++*), відкритою архітектурою та кодом, а також можливістю програмування через *USB* без додаткових програматорів.

Arduino дозволяє "перекинути міст" між віртуальним та фізичним світом. Пристрої на базі *Arduino* можуть збирати інформацію про навколишнє середовище за допомогою різноманітних датчиків (наприклад, температури, вологості, освітленості, біометричних показників) та керувати виконавчими механізмами (моторами, світлодіодами, реле).

Програмування мікроконтролера здійснюється за допомогою *Arduino IDE* (*Integrated Development Environment*), що базується на середовищі *Processing* та

використовує спрощену мову програмування *Wiring*. Пристрої *Arduino* можуть функціонувати автономно або взаємодіяти з програмним забезпеченням на комп'ютері (наприклад, з *Python, JavaScript* або спеціалізованими програмами візуалізації). Користувачі можуть як придбати готові плати, так і зібрати їх самостійно, оскільки все програмне забезпечення є безкоштовним, а оригінальні схеми (*CAD*-файли) доступні для вільного використання. Відкритість та доступність *Arduino* були відзначені багатьма нагородами, включаючи *Prix Ars Electronica*.

2.1 Апаратна платформа *Arduino*: Еволюція та різноманітність

Екосистема *Arduino* пропонує широкий спектр плат, що постійно еволюціонують для задоволення різноманітних потреб проєктів.

- *Arduino Uno*: Залишається однією з найпопулярніших та найпоширеніших плат. Вона базується на мікроконтролері *Atmel ATmega*328*P* і має стандартний *USB* порт для зручного підключення до комп'ютера. *Uno* є наступником *Duemilanove*, але використовує покращений чіп для *USB*-зв'язку та має більш зручне маркування виводів.

- *Arduino Leonardo*: Ця версія, на якій побудований наш пристрій, використовує мікроконтролер *ATmega*32*u*4. Її особливість у тому, що *ATmega*32*u*4 має вбудовану *USB*-комунікацію, що дозволяє *Arduino Leonardo* виступати як

клавіатура, миша або інший *HID*-пристрій для комп'ютера.

- *Arduino Mega* 2560: Одна з найпотужніших плат, заснована на мікроконтролері *ATmega*2560. Вона пропонує значно більше пам'яті та велику кількість вхідних/вихідних пінів, що робить її ідеальною для складних проєктів з великою кількістю датчиків та виконавчих пристроїв.

- *Arduino Due*: Значний крок вперед, оскільки це одна з перших плат *Arduino*, що використовує 32-бітний мікропроцесор *ARM Cortex-M*3 (*SAM*3*X*8*E*). Це забезпечує вищу швидкість обробки та більшу кількість пам'яті, відкриваючи нові можливості для ресурсоємних завдань.

- *Arduino Nano*: Дуже компактна версія, що ідеально підходить для інтеграції в обмежений простір або використання на макетних платах.

- *Arduino Micro*: Ще менша версія, схожа за функціоналом на *Leonardo*, але в мініатюрному форм-факторі.

- *Arduino Yún*/*Yun Rev*. 2: Поєднує в собі мікроконтролер *Arduino* (*ATmega*32*u*4) з *Linux*-процесором (*Atheros AR*9331), що дозволяє платі мати вбудовану підтримку *Wi-Fi*, *Ethernet* та запускати складніші мережеві програми.

- *Arduino MKR*-серія: Сучасна лінійка плат, розроблена для *IoT* (Інтернету речей) проєктів. Включає версії з вбудованим *Wi-Fi*, *Bluetooth*, *LoRa* та іншими бездротовими модулями. Приклади: *MKR WiFi* 1010, *MKR WAN* 1310.

- *Arduino Nano Every* / *Nano* 33 *IoT* / *Nano* 33 *BLE* / *Nano RP*2040 *Connect*: Новіші компактні плати *Nano*-серії, що пропонують покращену продуктивність та вбудовані комунікаційні можливості (*Wi-Fi*, *Bluetooth*).

- *Arduino Leonardo ETH* / *Uno WiFi Rev*. 2: Плати з вбудованими можливостями *Ethernet* та *Wi-Fi* відповідно, що спрощує підключення до мережі. - *Arduino Esplora* / *LilyPad* / *Fio* / *Mini*: Спеціалізовані плати для конкретних застосувань: *Esplora* – для інтерактивних проєктів, *LilyPad* – для "носимої" електроніки (можна вшивати в тканину), *Fio* – для бездротових проєктів (з роз'ємом для *XBee*), *Mini* – для мініатюрних рішень.

- *Arduino Pro*/*Pro Mini*: Розроблені для досвідчених користувачів, яким потрібні компактні та недорогі рішення для інтеграції в більші проєкти. Вони часто вимагають зовнішнього *USB*-адаптера для програмування.

Рисунок 2.2 - *Arduino Uno*

Однією з найпопулярніших і найдоступніших плат для початківців є *Arduino Uno* (рис 2.2).

2.2 Актуальні версії платформи *Arduino*: Коротка характеристика

Сучасна екосистема *Arduino* включає широкий спектр плат, кожна з яких оптимізована для певних завдань та рівнів досвіду:

- *Arduino Due*: Потужна плата на базі 32-бітного мікропроцесора *ARM Cortex M*3 (*ARM SAM*3*U*4*E*), що забезпечує високу продуктивність.
- *Arduino Leonardo*: Ця плата, на якій ґрунтується наш проєкт, використовує мікроконтролер *ATmega*32*u*4, що дозволяє їй працювати як *HID*-пристрій (миша, клавіатура) при підключенні до комп'ютера. Має компактний форм-фактор.
- *Arduino Yún* (та *Yun Rev*. 2): Інноваційна плата з вбудованою підтримкою *Wi Fi*, яка поєднує *ATmega*32*u*4 з процесором *Atheros AR*9331 (працює на *Linux*), що робить її ідеальною для *IoT*-проєктів.
- *Arduino Micro*: Дуже компактне рішення на базі *ATmega*32*u*4, ідеальне для проєктів з обмеженим простором.
- *Arduino Uno* (та *Uno WiFi Rev*. 2): Найпопулярніша базова платформа *Arduino*

з *USB*-підключенням. *Uno WiFi Rev*. 2 додатково має вбудований модуль *Wi Fi*.

- *Arduino Ethernet* (та *Ethernet Shield*): Плата з вбудованою мережевою підтримкою *Ethernet,* що дозволяє легко підключати пристрої до локальної мережі або Інтернету. Можливе живлення через *Ethernet* (*PoE*).

- *Arduino Nano* (та *Nano Every, Nano* 33 *IoT, Nano* 33 *BLE, Nano RP*2040 *Connect*): Широка лінійка компактних плат, що слугують для прототипування на макетних платах. Новіші версії пропонують *Wi-Fi, Bluetooth* та інші розширені можливості.

- *Arduino Mega ADK* (та *Mega* 2560): *Mega ADK* є версією *Mega* 2560 з підтримкою *USB*-хоста, що дозволяє взаємодіяти з *Android*-пристроями та іншими

USB-пристроями. *Mega* 2560 — це високопродуктивна плата з великою кількістю пінів та пам'яті.

- *Arduino LilyPad*: Спеціалізована фіолетова плата, призначена для носіння та інтеграції в тканини, ідеальна для проєктів "розумного одягу".

- *Arduino Fio*: Розроблена для бездротових застосувань, має роз'єм для модуля *XBee*, роз'єм для *LiPo*-акумулятора та вбудовану схему заряджання. - *Arduino Mini* / *Pro Mini*: Найменші та дуже компактні платформи, призначені для вбудованих проєктів, де простір є критичним. Часто вимагають зовнішнього *USB* адаптера для програмування.

Актуальні адаптери та аксесуари:

- *USB*-адаптери (наприклад, *USB-to-Serial Adapter*): Використовуються для підключення плат *Arduino* без власного *USB*-порту (наприклад, *Mini, Pro Mini, Ethernet*) до комп'ютера для програмування та обміну даними.

Arduino продовжує розвиватися, пропонуючи все нові рішення для різноманітних завдань, від простих прототипів до складних *IoT*-систем.

2.3 Плати розширення (*Shields*): Розширення функціональності *Arduino*

Плати розширення (*Shields*) – це спеціалізовані модулі, що встановлюються

поверх плат *Arduino*, значно розширюючи їх функціональність для керування різноманітними пристроями, збору даних та інтеграції в складніші системи. Вони спрощують розробку, дозволяючи додавати нові можливості без складного монтажу. Серед найпопулярніших та актуальних плат розширення:

- *Wi-Fi* та *Bluetooth Shields*: Забезпечують бездротове підключення до мереж за стандартами 802.11 *b*/*g*/*n* (*Wi-Fi*) або за протоколом *Bluetooth Low Energy* (*BLE*). Це дозволяє *Arduino* взаємодіяти з інтернетом, хмарними сервісами, смартфонами та іншими бездротовими пристроями, що є основою для проєктів Інтернету речей (*IoT*).

- *LoRa*/*LoRaWAN Shields*: Використовуються для далекосяжного бездротового зв'язку з низьким енергоспоживанням, ідеально підходять для *IoT* пристроїв, що працюють на великих відстанях з мінімальним споживанням батареї.

- *Motor Driver Shields*: Дозволяють *Arduino* керувати різними типами двигунів (постійного струму, кроковими, сервомоторами), а також зчитувати дані з енкодерів та датчиків положення, що необхідно для робототехніки та систем автоматизації.

- *Ethernet Shields*: Забезпечують дротове мережеве підключення (через *RJ*45 роз'єм), що дозволяє *Arduino* підключатися до локальних мереж та Інтернету для передачі даних та дистанційного керування.

- *Sensor Shields*: Надають зручні роз'єми та схеми для швидкого підключення великої кількості різноманітних датчиків (температури, вологості, освітлення, газу, руху тощо).

- *SD Card Shields*: Дозволяють зберігати великі обсяги даних на *SD*-картах, що корисно для реєстрації показань датчиків, логування подій або зберігання конфігураційних файлів.

- *Display Shields* (*LCD*/*OLED*): Надають можливість виводити текстову або графічну інформацію на невеликі екрани, що покращує користувацький інтерфейс проєктів.

2.4 *Arduino Leonardo*: Детальний огляд

є відмінним
і *USB*.

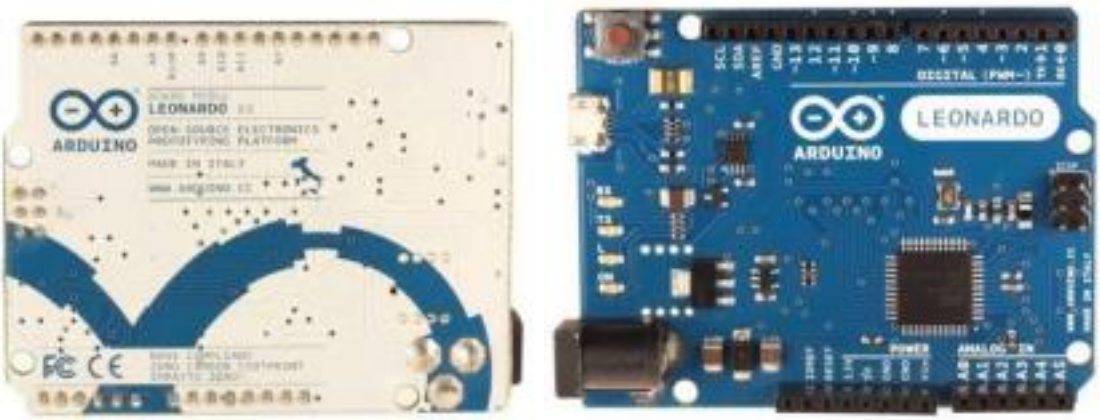

Рисунок 2.3 -

Arduino Leonardo

Загальна інформація

Arduino Leonardo – це мікроконтролерна плата, заснована на чіпі *ATmega*32*u*4. Вона пропонує 20 цифрових вхідних/вихідних пінів, з яких 7 можуть використовуватися для ШІМ (широтно-імпульсної модуляції), а 12 – як аналогові входи. Плата оснащена кварцовим резонатором на 16 МГц, портом *Micro-USB*, роз'ємом для зовнішнього живлення, портом *ICSP* (для внутрішньосхемного програмування) та кнопкою скидання. Для роботи *Leonardo* необхідно підключити її до комп'ютера через *USB*-кабель або забезпечити зовнішнє живлення від адаптера (змінного/постійного струму) чи акумулятора.

Ключова відмінність *Leonardo* від багатьох попередніх плат полягає в тому, що *ATmega*32*u*4 має вбудовану *USB*-комунікацію. Це означає, що *Arduino Leonardo* може бути розпізнана комп'ютером не тільки як віртуальний послідовний (*COM*) порт, але й як *USB*-клавіатура, миша або інший пристрій *Human Interface Device* (*HID*). Ця функція значно розширює можливості інтеграції та взаємодії з комп'ютером.

Особливості *Arduino Leonardo*

Таблиця 2.1 - Детальний опис ключових характеристик *Arduino Leonardo*

Особливість Значення	
Мікроконтролер	*ATmega*32*u*4

Робоча напруга	5 В
Вхідна напруга (рекомендована)	7-12 В
Вхідна напруга (гранична)	6-20 В
Цифрові Входи/Виходи	20 (7 з них підтримують ШІМ-вихід)
Аналогові Входи	12
Постійний струм на В/В піні	40 мА
Постійний струм на виводі 3.3 В	50 мА
Flash-пам'ять	32 КБ (з них 4 КБ зайнято завантажувачем)
SRAM (оперативна пам'ять)	2 КБ

Продовження таблиці 2.1

EEPROM (енергонезалежна пам'ять) 1 КБ

Тактова частота	16 МГц

Живлення

Arduino Leonardo може отримувати живлення як через *USB*-з'єднання, так і від зовнішнього джерела живлення. Вибір джерела живлення відбувається автоматично.

Зовнішнє живлення (не через *USB*) може подаватися через адаптер змінного/постійного струму (блок живлення) або від батареї. Адаптер живлення підключається через стандартний 2.1 мм роз'єм з позитивним центральним контактом. Батарея може бути підключена до пінів *Gnd* та *Vin* роз'єму живлення. Плата може працювати від зовнішнього джерела напругою від 6 В до 20 В. Однак слід бути обережними: якщо вхідна напруга зовнішнього джерела, що подається на

регулятор, буде менше 7 В, вихідна напруга 5 В може виявитися нижче необхідних 5 В, що призведе до нестабільної роботи плати. Використання напруги вище 12 В може спричинити перегрів вбудованого стабілізатора напруги та потенційне пошкодження плати. Рекомендований діапазон живлення становить від 7 В до 12 В.

Опис пінів живлення:

- *Vin*: Вхід для подачі нерегульованої напруги від зовнішнього джерела живлення, коли 5 В від *USB* або іншого стабілізованого джерела недоступні. - 5*V*: Регульоване джерело живлення 5 В, що використовується для живлення мікроконтролера та інших компонентів плати. Це живлення може надходити від піна *Vin* через стабілізатор, через *USB*-порт або від іншого регульованого джерела 5 В. - 3*V*3: Вихід 3.3 В, генерований вбудованим стабілізатором на платі. Максимальний струм, який можна зняти з цього піна, становить 50 мА. - *GND*: Піни заземлення.

- *IOREF*: Цей пін надає опорну напругу, з якою працюють входи/виходи плати. Для *Leonardo* це 5 В. Ця напруга використовується шилдами для коректного вибору їх робочої напруги.

Вхід та вихід

Кожен з 20 цифрових пінів *Leonardo* може бути налаштований як вхід або вихід за допомогою функцій *pinMode*(), *digitalWrite*() та *digitalRead*() у програмному коді. Всі піни працюють при напрузі 5 В. Кожен пін має внутрішній підтягувальний резистор (*pull-up resistor*) 20-50 кОм (за замовчуванням вимкнений) і може витримувати струм до 40 мА. Деякі піни мають спеціальні функції:

- Послідовний зв'язок (*Serial*): Піни 0 (*RX*) та 1 (*TX*). Ці піни використовуються для прийому (*RX*) та передачі (*TX*) даних через послідовний інтерфейс *TTL*. На *Leonardo,* вони підключені до відповідних пінів мікросхеми *ATmega*32*U*4, яка забезпечує *USB*-зв'язок. Важливо: клас *Serial* у *Arduino IDE* на *Leonardo* відноситься до *USB CDC* (*Communication Device Class*) послідовного з'єднання, тоді як послідовний зв'язок через піни 0 та 1 реалізується через клас *Serial*1.

- *I2C* (*TWI*): Піни 2 (*SDA*) та 3 (*SCL*). Ці піни використовуються для зв'язку за

протоколом *I2C* (*Two-Wire Interface*), що реалізується за допомогою бібліотеки *Wire*.

- Зовнішні переривання: Піни 2 та 3. Ці піни можуть бути налаштовані для спрацьовування переривань за низьким рівнем сигналу, наростаючим/спадаючим фронтом або зміною значення. Детальніше див. функцію *attachInterrupt*(). - ШІМ (*PWM*): Піни 3, 5, 6, 9, 10, 11 та 13. Будь-який з цих пінів може генерувати ШІМ-сигнал з 8-бітною роздільною здатністю за допомогою функції *analogWrite*().

- *SPI* (*Serial Peripheral Interface*): Розташований на роз'ємі *ICSP* (*In-Circuit Serial Programming*). Зв'язок *SPI* реалізується через ці піни за допомогою бібліотеки *SPI*. Важливо: на *Leonardo* піни *SPI* не виведені на стандартні цифрові піни, як у деяких інших версіях *Arduino*.

- *LED* (Вбудований світлодіод): Пін 13. До цього цифрового піна підключений вбудований світлодіод. Коли на пін 13 подається високий рівень сигналу, світлодіод вмикається.

- Аналогові входи: *A*0-*A*5, а також *A*6-*A*11 (що знаходяться на цифрових пінах 4, 6, 8, 9, 10 та 12). *Leonardo* має загалом 12 аналогових входів, позначених від *A*0 до *A*11. Усі аналогові входи можуть також функціонувати як цифрові входи/виходи. Входи *A*0-*A*5 ідентичні аналоговим входам *Arduino Uno*. Роздільна здатність аналогових входів становить 10 біт, що дозволяє розрізняти 1024 різні значення. За замовчуванням аналогові входи вимірюють напругу від 0 В (земля) до 5 В, проте верхню межу діапазону можна змінити за допомогою піна *AREF* та функції *analogReference*().

Додаткові службові піни:

- *AREF*: Опорна напруга для аналогових входів. Використовується разом з функцією *analogReference*() для зміни діапазону аналогових вимірювань. - *Reset*: Низький рівень сигналу на цьому піні перезавантажує мікроконтролер. Цей пін часто використовується для підключення кнопки скидання на платах розширення, що дозволяє уникнути доступу до основної кнопки скидання на самій платі *Arduino*.

Комунікація

Arduino Leonardo пропонує кілька способів взаємодії з комп'ютером, іншими

платами *Arduino* або іншими мікроконтролерами:

- Послідовний інтерфейс (*UART*): Мікроконтролер *ATmega*32*U*4 підтримує послідовний інтерфейс *UART TTL* (5В), доступний на пінах 0 (*RX*) та 1 (*TX*). - *USB CDC* (*Virtual COM Port*): *ATmega*32*U*4 також дозволяє встановити послідовний зв'язок через *USB* з програмами на комп'ютері, що дозволяє їм "спілкуватися" з платою через віртуальний *COM*-порт. *Leonardo* підключається як *USB* 2.0 пристрій, і для *Windows* може знадобитися *.inf* файл для драйвера. Вбудований "Послідовний монітор" в *Arduino IDE* дозволяє легко надсилати та отримувати текстові дані. Світлодіоди *RX* та *TX* на платі блиматимуть під час передачі даних через *USB* (але не при використанні послідовного зв'язку через піни 0 та 1). - *SoftwareSerial*: Використовуючи бібліотеку *SoftwareSerial*, можна реалізувати додаткові послідовні з'єднання через будь-який цифровий пін *Leonardo*.

- *I2C* (*TWI*) та *SPI*: *ATmega*32*U*4 підтримує інтерфейси *I2C* (*TWI*) та *SPI*. Для зручного використання шини *I2C* в *Arduino* є бібліотека *Wire*, а для *SPI* – бібліотека *SPI*.

- Емуляція *HID* (*Human Interface Device*): При підключенні до комп'ютера *Arduino Leonardo* може ідентифікуватися як *USB*-миша або *USB*-клавіатура. Це контролюється за допомогою вбудованих класів *Keyboard* та *Mouse* в *Arduino IDE*.

Програмування

Програмування плати *Arduino Leonardo* здійснюється за допомогою *Arduino IDE*. У меню "Інструменти" -> "Плата" необхідно обрати "*Arduino Leonardo*" відповідно до використовуваного мікроконтролера.

Мікроконтролер *ATmega*32*U*4 на *Leonardo* постачається з попередньо записаним завантажувачем (*bootloader*), що дозволяє завантажувати нові програми без необхідності використання зовнішнього програматора. Зв'язок для завантаження відбувається за протоколом *AVR*109. Однак, за бажанням, мікроконтролер може бути запрограмований безпосередньо через піни *ICSP* (*In-Circuit Serial Programming*).

Автоматичний (*Soft*) перезапуск

Однією з особливостей *Leonardo* є автоматичний перезапуск (*soft reset*), який виконується програмою *Arduino IDE* на комп'ютері перед завантаженням нового коду, замість ручного натискання кнопки на платі. Перезапуск ініціюється, коли віртуальний *COM*-порт *CDC* відкривається та закривається на швидкості 1200 бод. У цей момент мікроконтролер перезавантажується, і *USB*-з'єднання тимчасово розривається. Після перезавантаження активується завантажувач, який залишається активним приблизно 8 секунд, очікуючи на завантаження нового коду. Ви також можете ініціювати завантажувач, натиснувши фізичну кнопку "*Reset*" на платі. Важливо зазначити, що після подачі живлення контролер одразу запускає завантажену користувацьку програму без необхідності активації завантажувача, якщо не очікується нове завантаження.

Захист *USB*-роз'єму від перевантаження

Arduino Leonardo має вбудований самовідновлюваний запобіжник, який захищає *USB*-порт комп'ютера від коротких замикань та перевантажень по струму.

Хоча більшість комп'ютерів вже мають власні схеми захисту, цей запобіжник забезпечує додатковий рівень безпеки. Якщо струм, що проходить через *USB*-порт, перевищує 500 мА, запобіжник спрацьовує, розриваючи ланцюг, доки струм не повернеться до нормального рівня, захищаючи таким чином як плату, так і *USB*-порт комп'ютера.

Фізичні характеристики

Розміри друкованої плати *Leonardo* становлять приблизно 6.9 *x* 5.3 см. Важливо зазначити, що *USB*-роз'єм та роз'єм живлення виступають за ці габарити. Плата має чотири монтажні отвори, що дозволяють надійно закріпити її на плоскій поверхні. Відстань між цифровими пінами 7 та 8 становить 0.4 см, тоді як відстань між іншими пінами складає 0.25 см.

2.5 Розробка програмного забезпечення для мікроконтролерів: Збір та обробка ЕКГ-даних

Для візуалізації графіка серцевого ритму нам потрібно перетворити наш

пристрій на своєрідний цифровий осцилограф. Мікроконтролер *Arduino* виступатиме в ролі пристрою для вимірювання та оцифрування аналогового сигналу.

Щоб отримати достатньо чіткий і детальний графік серцевої активності, нам необхідно виконувати вимірювання з високою частотою, приблизно 1000 разів за секунду (1 кГц). *Arduino* здатний виконувати операції зчитування значно швидше – кожні 100 мікросекунд, що дозволяє робити до 10 000 вимірювань за секунду. Ця швидкість з запасом задовольняє наші потреби.

Ключовим аспектом для коректного аналізу даних є рівномірність інтервалів між вимірюваннями. Для цього ми будемо використовувати внутрішню оперативну пам'ять (*SRAM*) *Arduino* як буфер. *Arduino Leonardo* має 2 КБ *SRAM*. Це дозволяє нам зберігати близько 1000 цілочисельних значень (*int*), кожне з яких займає 2 байти пам'яті, що становить приблизно 80% доступної *SRAM* (1000 * 2 байти = 2000 байт = 1.95 КБ). Таким чином, ми можемо накопичувати дані протягом однієї секунди, а потім передавати весь буфер даних на комп'ютер раз на секунду.

Чому не передавати дані одразу після кожного вимірювання? Пряма передача даних через *USB*-порт після кожного вимірювання виявиться неефективною. Кожна операція передачі вимагає певного часу на відкриття порту та встановлення з'єднання, що призведе до нерівномірного надходження даних і зниження фактичної частоти дискретизації. Натомість, накопичення даних у внутрішньому буфері забезпечує їх збір з точно однаковим інтервалом, обмеженим лише апаратною швидкістю мікроконтролера. Після заповнення буфера весь пакет даних ефективно передається на ПК для подальшої обробки та візуалізації.

Для реалізації описаної функціональності було розроблено наступний простий скетч (програму) для *Arduino*:

```
int values[1000]; // Масив для зберігання 1000 цілочисельних значень
int count = 0; // Лічильник кількості зібраних вимірювань
void setup() {
 Serial.begin(9600); // Ініціалізація послідовного порту зі швидкістю 9600 бод
 }
```

```
void loop() {
 if (count < 1000) {
 // Якщо буфер не заповнений
 values[count] = analogRead(A1); // Зчитуємо аналогове значення з піна A1
count++; // Збільшуємо лічильник
 delay(1); // Затримка на 1 мілісекунду (для 1000 вимірювань/сек)  } else {
 // Якщо буфер заповнений (зібрано 1000 вимірювань)
 for (int i = 0; i < count; i++) {
 Serial.print(values[i]); // Виводимо кожне значення з буфера
 Serial.print(","); // Додаємо кому як роздільник
 }
 Serial.println(); // Перехід на новий рядок після виведення всіх даних  count
= 0; // Скидаємо лічильник для нового циклу збору  }
```



```
}
```

Пояснення ключових елементів коду:

- *void setup*(): Цей блок коду виконується один раз при запуску або перезавантаженні *Arduino*. Він використовується для ініціалізації налаштувань. - *Serial.begin*(9600);: Встановлює швидкість обміну даними через послідовний порт (*USB*) на 9600 бод (біт за секунду). Ця швидкість є стандартною, але може бути збільшена для швидшої передачі даних (наприклад, 115200) залежно від потреб проєкту та стабільності зв'язку. Допустимі значення: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200.

- *void loop*(): Цей блок коду виконується нескінченно після завершення *setup*(). Він є основним циклом програми, де відбувається збір та обробка даних. - *if* (*count* < 1000): Перевіряє, чи буфер *values* ще не заповнений. - *values*[*count*] = *analogRead*(*A*1);: Зчитує аналогове значення з аналогового входу *A*1 (оскільки аналогові входи *Arduino Leonardo A*6-*A*11 знаходяться на цифрових пінах, *A*1 є традиційним аналоговим входом) і зберігає його у масиві *values*. Функція *analogRead*() повертає 10-бітне значення (від 0 до 1023), що відповідає напрузі від 0

В до 5 В.

- *count*++;: Збільшує лічильник зібраних значень.

- *delay*(1);: Зупиняє виконання програми на 1 мілісекунду. Це дозволяє досягти частоти дискретизації приблизно 1000 вимірювань за секунду (оскільки 1000 мс = 1 секунда).

- *else*: Якщо буфер *values* заповнений (тобто *count* досяг 1000). - *for* (*int i* = 0; *i* < *count*; *i*++) { *Serial*.*print*(*values*[*i*]); *Serial*.*print*(","); }: Проходить по всьому буферу *values* і виводить кожне зібране значення на послідовний порт. Для зручності подальшої обробки на ПК, значення розділяються комою (*CSV*-формат).

- *Serial*.*println*();: Після виведення всіх значень буфера, додає символ нового рядка, що дозволяє легко розділяти пакети даних на стороні ПК. - *count* = 0;: Скидає лічильник, щоб почати збір нового пакету даних.

- *Serial*.*print*() / *Serial*.*println*(): Ці функції використовуються для виведення даних на послідовний порт. Дані перетворюються в кодування *ASCII* і надсилаються байт за байтом. *Serial*.*println*() додатково додає символи нового рядка після виведених даних.

Цей підхід дозволяє ефективно збирати високочастотні дані з аналогових датчиків та передавати їх на комп'ютер для подальшої візуалізації та аналізу, наприклад, для створення графіка ЕКГ у реальному часі.

РОЗДІЛ 3
ПРОЄКТУВАННЯ АНАЛОГОВОЇ ЧАСТИНИ

Тепер ми переходимо до розробки аналогової частини нашого пристрою, яка є критично важливою для захоплення та підготовки слабких біоелектричних сигналів серця. Спочатку ми вивчили технічну документацію на інструментальний операційний підсилювач *AD*620, яка надала базове розуміння його роботи.

Приклад типової схеми підключення інструментального операційного

підсилювача з технічної документації, яка показує його основні компоненти та принцип дії можна побачити на рисунку 3.1.

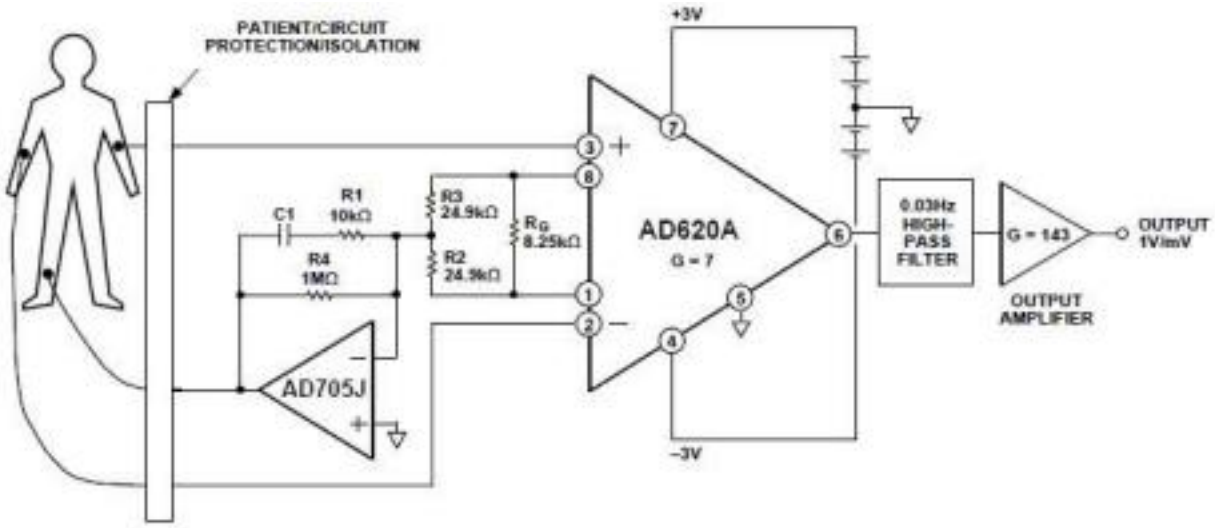

Рисунок 3.1 - Типова принципова схема інструментального підсилювача

Електрокардіограф (ЕКГ) реєструє різницю електричних потенціалів між двома точками на тілі людини, що виникає внаслідок роботи міокарда (серцевого м'яза). Найбільш сильні сигнали можна отримати, розмістивши електроди на внутрішній стороні лівого та правого зап'ясть або на внутрішній стороні ліктів. У цих ділянках проходять великі артерії, які слугують провідниками біострумів серця, і вони відділені від зовнішнього середовища лише тонким шаром шкіри, що забезпечує хороший контакт.

Для запису різниці потенціалів між двома сигналами використовується диференціальний підсилювач. Його модифікований та покращений варіант, що

забезпечує високу точність та великий вхідний опір, називається інструментальним підсилювачем.

Зазвичай інструментальні підсилювачі складаються з трьох операційних підсилювачів (ОП). На схемі (Рисунок 3.2, що буде нижче) ОП *DA*3 з резисторами *R*1, *R*2, *R*3, *R*4 фактично є класичним диференціальним підсилювачем. А ОП *DA*1 та *DA*2 слугують для збільшення вхідного опору всього підсилювача. Це дозволяє використовувати лише один резистор *Rg* для зміни коефіцієнта посилення всієї схеми, не порушуючи при цьому симетрії плечей диференціального підсилювача та зберігаючи високий коефіцієнт придушення синфазних перешкод.

Узагальнена принципова схема типового інструментального підсилювача, що ілюструє його структуру з трьома операційними підсилювачами зображена на рисунку 3.2.

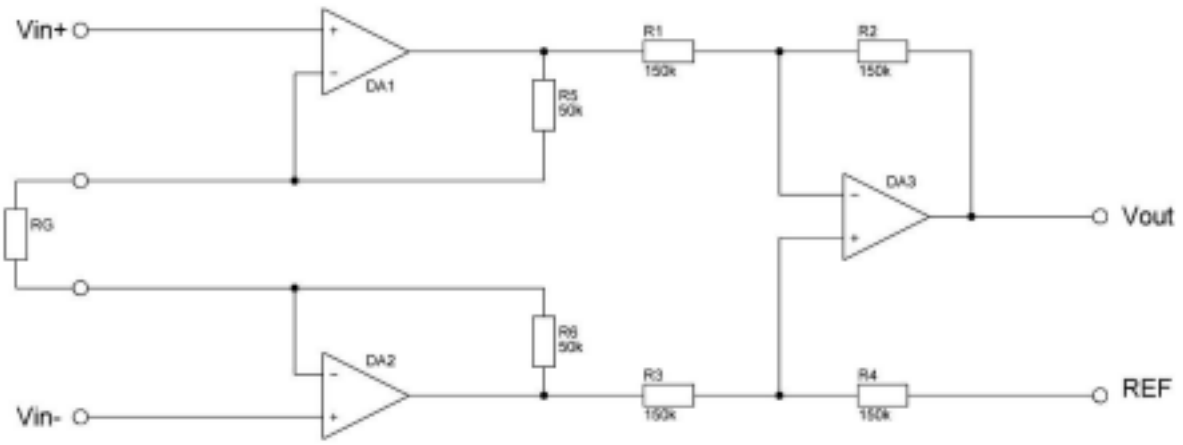

Рисунок 3.2 - Принципова схема інструментального підсилювача

У нашому проєкті схема реалізована з використанням сучасної мікросхеми *INA*333 від *Texas Instruments*. Ця мікросхема є мікропотужним (всього 50 мкА), інструментальним підсилювачем з нульовим дрейфом (що забезпечує високу стабільність) та рейка-до-рейка (*rail-to-rail*) вихідним сигналом, що дозволяє використовувати повний діапазон напруги живлення для вихідного сигналу. Мікросхема *INA*333, окрім диференціальних входів, також має вивід *REF* (*Reference*) та вивід для підключення зовнішнього резистора *Rg* (для налаштування коефіцієнта посилення). Для двополярної схеми живлення вивід *REF* зазвичай

підключається до загальної точки схеми. Однак, оскільки ми плануємо використовувати підсилювач з мікроконтролером типу *Arduino,* який має однополярний аналоговий вхід АЦП (від 0 В до +5 В), а також вихід +5 В для живлення зовнішніх пристроїв, було прийнято рішення побудувати схему однополярного живлення від +5 В.

У цьому випадку вивід *REF* необхідно підключити до опорної напруги, що дорівнює половині напруги живлення (тобто 2.5 В). Найпростіша реалізація джерела опорної напруги – це використання дільника напруги, що складається з двох резисторів однакового номіналу.

Проблема дільника напруги та її рішення

Використання простого дільника напруги для формування опорної напруги на виводі *REF*, підключеного до *INA*333, може мати недоліки. Якщо резистори дільника мають високий опір, це може вплинути на симетрію диференціального каскаду всередині *INA*333 (особливо якщо один з резисторів дільника додається до резистора *R*4 з вищезгаданої класичної схеми), що знизить коефіцієнт придушення синфазного шуму (*CMRR*) та точність посилення. Це відбувається через порушення симетрії гілок інструментального підсилювача. Крім того, температурний дрейф зовнішніх резисторів може відрізнятися від вбудованих, погіршуючи стабільність. Використання низькоомного дільника напруги, у свою чергу, призведе до значного збільшення струму споживання, що є неприйнятним для пристроїв з автономним живленням.

Цю проблему можна ефективно вирішити за допомогою повторювача напруги на основі операційного підсилювача.

Схема, яка показує, як можна сформувати стабільну опорну напругу за допомогою повторювача напруги зображена на рисунку 3.3.

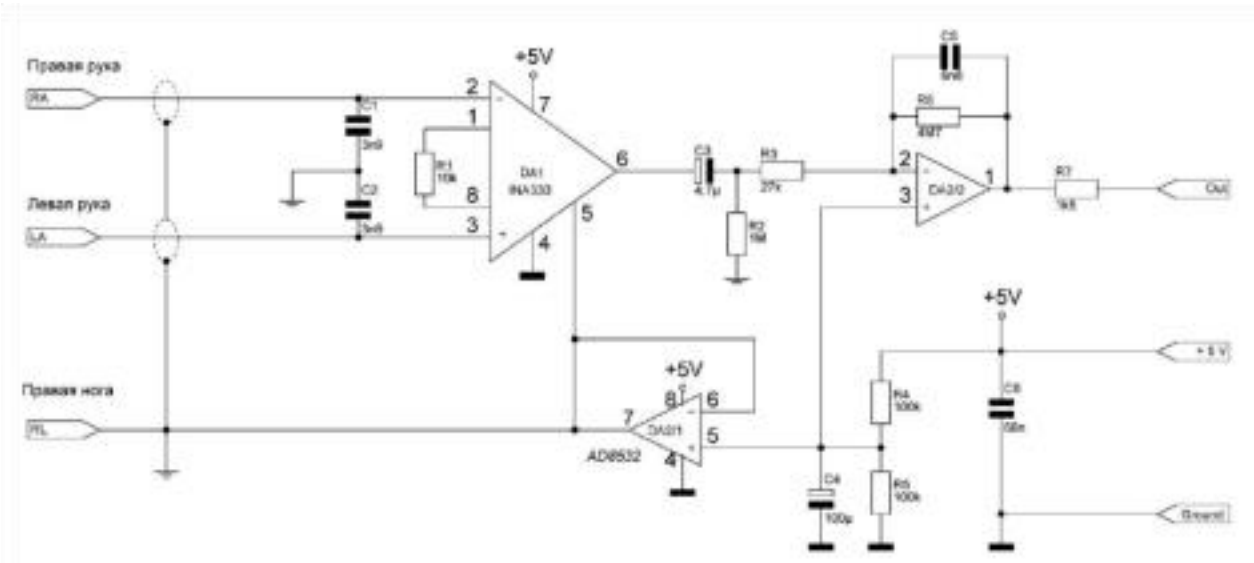

Рисунок 3.3 - Формування опорної напруги з буфером

Використання достатньо великого конденсатора (наприклад, *C* = 100 мкФ) на виході повторювача напруги значно зменшує вплив пульсацій джерела живлення на підсилювач та підтримує високий коефіцієнт придушення пульсацій живлення (близько 100 дБ). Недоліком цієї схеми є відносно тривалий час встановлення опорної напруги при запуску. Однак для ЕКГ це не критично, оскільки пацієнт

повинен залишатися спокійним протягом певного періоду (близько 10 секунд) для стабілізації сигналу. Якщо потрібне швидше увімкнення схеми вимірювання, повторювач напруги можна реалізувати на основі активного фільтра другого або вищого порядку, що дозволить використовувати менші ємності конденсаторів для прискорення стабілізації.

Коефіцієнт посилення та фільтрація

Рівень біосигналу, знятого з тіла людини, зазвичай знаходиться в межах 1 мілівольта (мВ). Щоб подати його на вхід АЦП *Arduino*, який працює з напругою від 0 В до 5 В, сигнал потрібно посилити як мінімум у 1000 разів (до 1 вольта, щоб мати запас для зміни). Однак, якщо застосувати такий високий коефіцієнт посилення одразу в першому каскаді, ми посилимо і постійну складову сигналу (яка є значно більшою за змінну) у 1000 разів, що призведе до перевантаження виходу підсилювача.

Тому коефіцієнт посилення першого каскаду (інструментального підсилювача) був обраний відносно невеликим – близько 10 разів. Він встановлюється зовнішнім резистором *Rg*, номінал якого визначається згідно з документацією на мікросхему *INA*333.

Основне посилення, приблизно в 100 разів, виконується другим каскадом підсилювача. Але перед цим, за допомогою розділяючого конденсатора (*C*3), постійна складова сигналу ізолюється. Це дозволяє підсилювати лише змінну складову ЕКГ сигналу. Конденсатор *C*5 використовується для зменшення посилення високочастотних перешкод, які можуть потрапити на високоомний вхід підсилювача, виконуючи роль фільтра низьких частот.

Оскільки АЦП *Arduino* працює в діапазоні від 0 В до 5 В, вихідний підсилювач також застосовує зміщення (*offset*) на 2.5 В. Тобто, за відсутності вхідного сигналу, на вхід АЦП буде подаватися постійна напруга 2.5 В. Змінна складова ЕКГ-сигналу буде змінюватися в межах ±2.5 В відносно цієї постійної складової, дозволяючи повністю використовувати динамічний діапазон АЦП.

Для зменшення рівня перешкод на вході підсилювача рекомендується використовувати короткі екрановані кабелі для підключення електродів, довжиною не більше 1 метра. Якщо потрібно використовувати довші кабелі, необхідно

розмістити буферні повторювачі напруги безпосередньо біля вхідних контактів електродів. Таким чином, джерело сигналу для інструментального підсилювача стане низькоомним, а рівень наведених перешкод буде мінімізований. Однак, у цьому випадку необхідно буде також прокласти додатковий кабель для живлення цих повторювачів.

Кінцева принципова схема аналогового підсилювача, що включає всі вищезгадані компоненти та рішення зображена на рисунку 3.4.

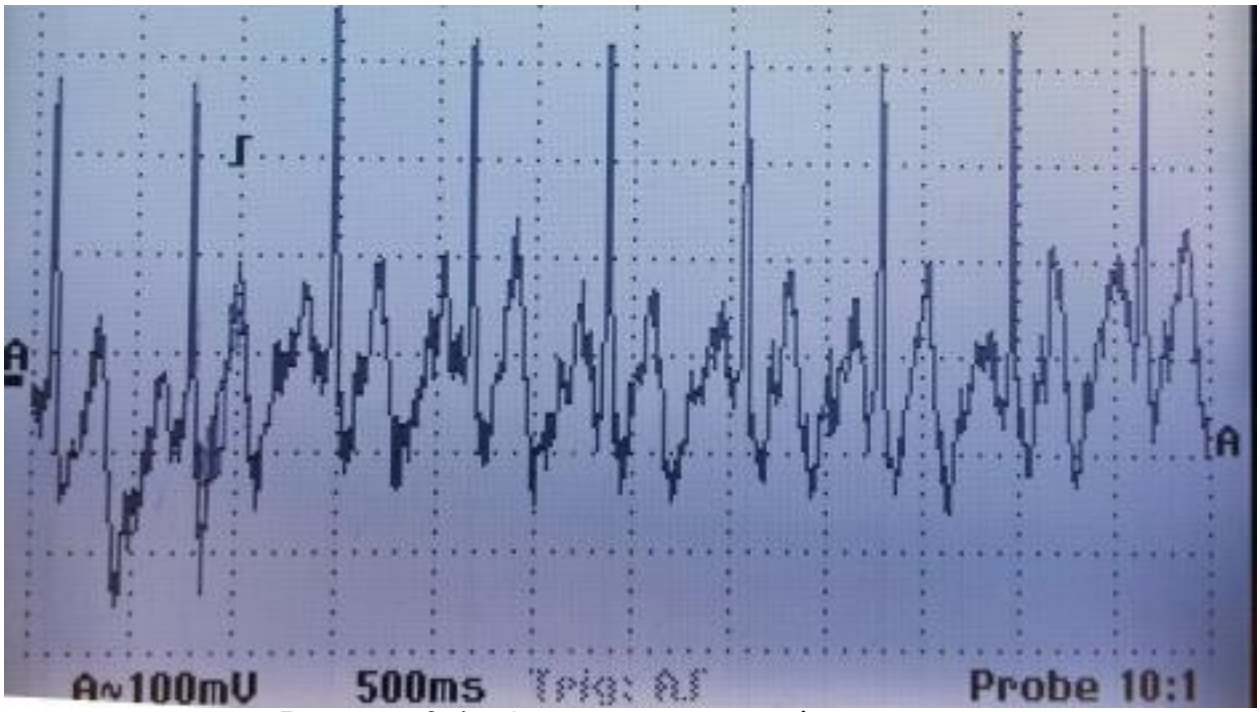

Рисунок 3.4 - Остаточна схема підсилювача

Після реалізації та тестування описаного підсилювача, ми отримали чіткі ЕКГ сигнали. Приклад того, як це виглядає на екрані цифрового осцилографа можна побачити на рисунку 3.5.

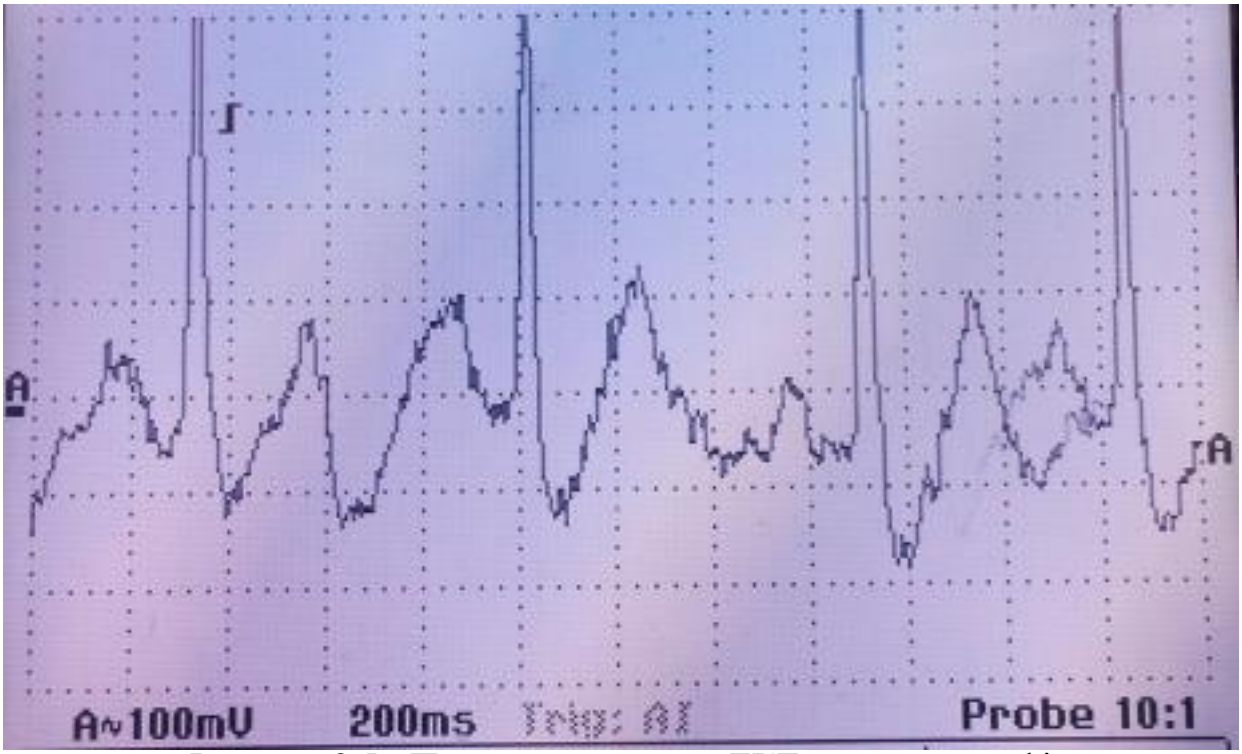

Рисунок 3.5 - Приклад отриманої ЕКГ на осцилографі

Цифрова обробка сигналів (ЦОС) – це галузь комп'ютерних наук та інженерії, що стрімко розвивається, охоплюючи як апаратні, так і програмні аспекти. Вона тісно пов'язана з теорією інформації, зокрема з теорією оптимального прийому сигналу, та теорією розпізнавання образів. Основне завдання ЦОС – виділення корисного сигналу від шуму та перешкод різного походження, а також автоматична класифікація та ідентифікація сигналів.

ЦОС оперує числовими послідовностями або символами для представлення сигналів. Її мета може бути різною: від оцінки характерних параметрів сигналу до перетворення сигналу в більш зручну для використання форму. Класичні методи чисельного аналізу, такі як інтерполяція, інтегрування та диференціювання, є фундаментальними алгоритмами ЦОС. Поява високошвидкісних цифрових комп'ютерів сприяла розробці все більш складних та інтелектуальних алгоритмів обробки сигналів. Сучасний прогрес у технологіях інтегральних схем дозволяє ефективно створювати надзвичайно потужні та складні системи ЦОС, які раніше були немислимі через високу вартість або технічні обмеження.

ЦОС широко використовується в безлічі галузей, включаючи біомедицину (наприклад, аналіз ЕКГ, ЕЕГ), акустику, радіолокацію, сейсмологію, телекомунікації, системи передачі даних, ядерну техніку та багато інших.

Методи ЦОС (*DSP*), які виконують обчислювальні операції з цифровими сигналами. Ці алгоритми включають:

- Цифрову фільтрацію: Видалення небажаних компонентів (шумів) або виділення певних частотних діапазонів.
- Спектральний аналіз: Визначення частотного складу сигналу (за допомогою ШПФ).
- Кореляційний аналіз: Вимірювання схожості між сигналами. - Модуляція та демодуляція сигналів: Процеси, що використовуються у зв'язку для передачі інформації.
- Адаптивна обробка: Алгоритми, які автоматично підлаштовуються під змінювані умови сигналу або середовища.

3.1 Аналого-цифрове перетворення (АЦП) та Цифро-аналогове перетворення (ЦАП)

Аналого-цифрове перетворення (АЦП) та цифро-аналогове перетворення (ЦАП) є фундаментальними концепціями в електроніці, оскільки більшість реальних пристроїв взаємодіють з аналоговими сигналами, тоді як комп'ютери та цифрові системи можуть обробляти лише цифрові сигнали. Розуміння цих процесів є ключовим для інтеграції фізичного світу з обчислювальними системами.

3.1.1 Типи сигналів

Перш ніж зануритися в сам процес перетворення, важливо розуміти, які типи сигналів існують:

- Аналогові сигнали: Є безперервними в часі; вони визначені в усі моменти часу і можуть набувати будь-яких значень у певному діапазоні. Приклади: звук, температура, тиск.
- Дискретні сигнали: Це сигнали, представлені послідовністю відліків, тобто значеннями сигналу, виміряними в дискретні (окремі) моменти часу. Наприклад, вимірювання температури кожну секунду.

- Цифрові сигнали: Це сигнали, які є дискретними як у часі (або просторі), так і квантованими за рівнем. Значення можуть приймати лише певні, фіксовані рівні. Усі обчислювальні процеси в комп'ютерах базуються саме на цифрових сигналах (представлених бінарним кодом).

Блок-схема процесу цифрової обробки сигналів зображена на рисунку 3.6.

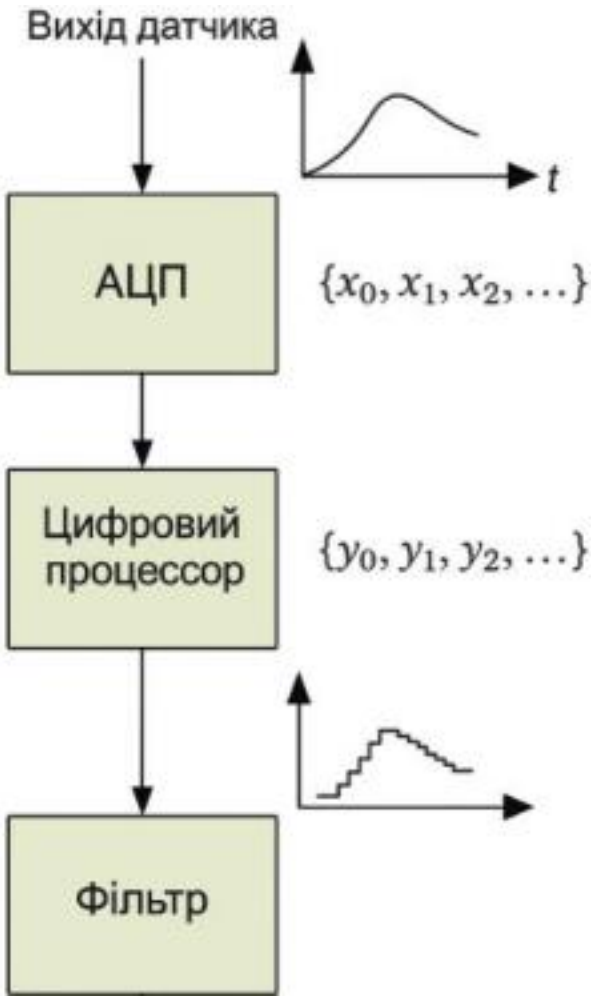

Рисунок 3.6 - Блок-схема цифрової обробки сигналів

Схематична блок-схему цифрової обробки сигналів з наступними основними блоками, розташованими послідовно:

1. Аналоговий сигнал (вхід)

2. Аналого-цифровий перетворювач (АЦП)

3. Цифровий сигнал (виміряні дискретні та квантовані значення) 4. Процесор цифрових сигналів (*DSP*) / Мікроконтролер / Комп'ютер (блок, де відбувається обробка: фільтрація, аналіз, алгоритми тощо)

5. Цифро-аналоговий перетворювач (ЦАП)

6. Аналоговий сигнал (вихід, наприклад, для керування пристроєм або відтворення звуку)

3.1.2 Аналого-цифрове перетворення сигналу (АЦП)

Аналоговий сигнал перетворюється на дискретний, тобто знімаються його значення з певним періодом дискретизації (*T*).

Частота дискретизації (*fs*) визначається як обернена величина до періоду дискретизації:

$$\mathrm{f}_s = \frac{1}{T} \quad (3.1)$$

Важливо, щоб процес отримання кожного відліку вхідного сигналу займав дуже малу частину періоду дискретизації. Це допомагає мінімізувати помилки динамічного перетворення, які можуть виникнути через зміни сигналу протягом самого часу вибірки.

Вибір оптимальної частоти дискретизації керується Теоремою Котельникова (або Найквіста-Шеннона). Ця теорема стверджує, що для точної реконструкції безперервного сигналу за його дискретними відліками частота дискретизації (*fs*) повинна бути щонайменше вдвічі більшою за максимальну частоту (*fmax*):

$$\mathrm{f}_s \geq 2 \cdot \mathrm{f}_{\max} \quad (3.2)$$

Якщо ця умова не виконується, виникає явище аліасингу (накладання спектрів), коли високочастотні компоненти сигналу помилково інтерпретуються як низькочастотні, що призводить до спотворення відновленого сигналу. Спектральне представлення сигналів

Будь-який сигнал можна представити у спектральній формі, як суму (або інтеграл) гармонічних складових (синусів і косинусів) з певними амплітудами та фазами. Це концепція перетворення Фур'є.

- Для періодичних сигналів це представлення має вигляд суми (ряд Фур'є). - Для неперіодичних сигналів це представлення має вигляд інтеграла (інтеграл Фур'є).

Перехід до спектрального представлення здійснюється за допомогою прямого перетворення Фур'є.

Приклад, що ілюструє перехід від періодичної функції до її спектрального представлення зображено на рисунку 3.7:

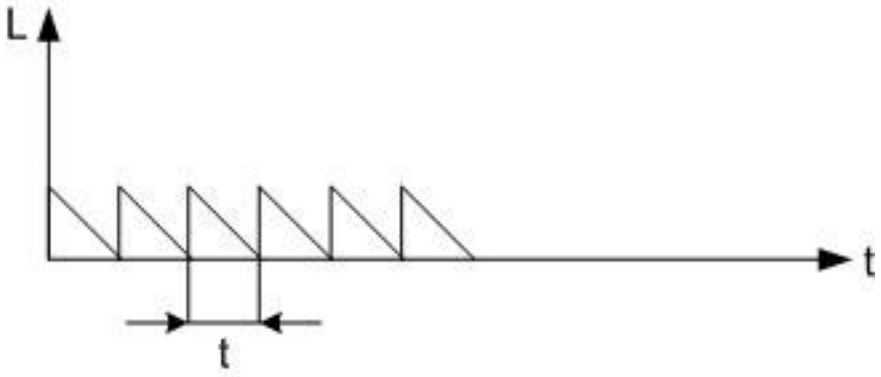

Рисунок 3.7 - Перехід до спектрального представлення (оновлено)

Загальновідомо, що періодичні функції, які відповідають умовам Діріхле (досить "добре поводяться"), можуть бути представлені рядом гармонічних функцій (рядом Фур'є):

$$x(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos(\omega_n t + \phi_n) \tag{3.3}$$

Згідно з формулою Ейлера, будь-яку гармонічну функцію можна представити в комплексній експоненційній формі:

(3.4)

Тоді ряд Фур'є можна записати у комплексній формі:

(3.5)

Де:

- $f1$ — частота першої гармоніки.
- $fn=n \cdot f1$ — частота n-ї гармоніки.
- $\omega n=2\pi fn$ — кругова частота n-ї гармоніки.
- $Cn=|Cn|ej\varphi n$ — комплексна амплітуда n-ї гармоніки.
- $|Cn|$ — амплітудний спектр (сукупність амплітуд усіх гармонік).

- φn — фазовий спектр (сукупність фаз усіх гармонік).

Приклад, що демонструє амплітудний спектр певної функції представлено на рисунку 3.8.

Рисунок 3.8 - Приклад амплітудного спектру

Для неперіодичних функцій період (T) прямує до нескінченності ($T \to \infty$), а частота першої гармоніки ($f1$) прямує до нуля ($f1 \to 0$). У цьому випадку сума в ряді Фур'є замінюється інтегралом, а дискретні частоти — безперервно змінною частотою.

Пряме перетворення Фур'є неперіодичних сигналів (інтеграл Фур'є):

(3.6)

Тобто, спектр неперіодичної функції представлений сумою нескінченної кількості гармонічних коливань, частоти яких нескінченно близькі одна до одної, формуючи безперервний спектр.

3.1.3 Квантування сигналу

Це процес, при якому безперервний діапазон значень амплітуди дискретного

сигналу перетворюється на обмежений набір дискретних рівнів. Кожному відліку сигналу присвоюється найближче значення з цього набору рівнів.

Графічне представлення процесу квантування зображено на рисунку 3.9.

Рисунок 3.9 - Квантування сигналу

Графік відображає:

1. Безперервну аналогову хвилю (наприклад, синусоїду).

2. Дискретні відліки.

3. Прямі, що представляють дозволені рівні квантування.

4. Ступінчасту "цифрову" апроксимацію аналогової хвилі, де кожен відлік "притягується" до найближчого рівня квантування.

Кількість рівнів квантування (N) визначається за формулою:

(3.7)

Де:

- n — розрядність АЦП (кількість бітів, що використовуються для представлення кожного відліку).

- N — кількість квантованих рівнів.

Наприклад, якщо АЦП має 10-бітну розрядність (як в *Arduino*), то N=210=1024 рівні.

Вибір кількості рівнів квантування сигналу – це завжди компроміс. З одного боку, для достатньо точного представлення сигналу та мінімізації помилки

квантування (різниці між фактичним значенням сигналу та його квантованим представленням) потрібна велика кількість рівнів квантування (тобто вища розрядність АЦП). З іншого боку, збільшення кількості рівнів (розрядності коду)

призводить до збільшення обчислювальних ресурсів (пам'яті, часу обробки) та складності апаратної реалізації. Оптимальний вибір залежить від конкретних вимог до точності та доступних ресурсів системи.

3.2 Інтерполяційний цифровий фільтр: Покращення якості сигналу

Інтерполяція в цифровій обробці сигналів – це процес збільшення кількості відліків сигналу за одиницю часу, що призводить до підвищення частоти дискретизації. Коефіцієнт інтерполяції – це відношення нової частоти дискретизації до початкової. Зазвичай це ціле число.

Згідно з теоремою Котельникова (Найквіста-Шеннона), збільшення частоти дискретизації сигналу розширює смугу частот, яку ці відліки можуть коректно описувати. У ширшу смугу частот потрапляють множинні "дзеркальні" копії (або "образи") спектра вихідного сигналу. При інтерполяції нашим завданням є вибрати потрібний образ спектра (зазвичай той, що знаходиться в діапазоні від 0 Гц до початкової верхньої частоти сигналу, *fV*) та придушити всі інші.

Це завдання вирішується за допомогою цифрового фільтра, який називається інтерполяційним фільтром. Цей фільтр обчислює значення сигналу між вихідними відліками, ефективно "заповнюючи прогалини" та згладжуючи сигнал. Давайте розглянемо цей процес на прикладі (рис. 3.10).

Рисунок 3.10 - Приклад інтерполяції: Покроковий процес

Щоб інтерполювати цей сигнал, ми збільшуємо кількість відліків за одиницю часу. На початковому етапі інтерполяції, нові проміжні відліки заповнюються нульовими значеннями. Це ефективно "розтягує" сигнал у часі, створюючи порожнечі між вихідними відліками (рис. 3.11).

Рисунок 3.11 - Сигнал на вході інтерполяційного фільтра

Тепер розглянемо спектр цього сигналу. Вставка нулів призводить до появи повторюваних копій (образів) початкового спектра сигналу на вищих частотах.

Рисунок 3.12 -

Частотний спектр сигналу після вставки нулів

Щоб інтерполювати сигнал, нам потрібно придушити ці небажані високочастотні спектральні образи, залишивши лише основний (від 0 до *fV*). Для цього використовується цифровий фільтр, який є фільтром низьких частот.

Ми прагнемо досягти високого рівня ослаблення небажаних спектральних компонентів, наприклад, -75 дБ. Такі параметри можна реалізувати за допомогою цифрового КІХ-фільтра (*FIR*-фільтра) з великою кількістю "відводів" (*taps*), наприклад, 128 відводів (рис 3.13).

Рисунок 3.13 - Амплітудно-частотна характеристика (АЧХ) інтерполяційного КІХ-фільтра

Графік АЧХ цифрового фільтра, показує:

1. Смугу пропускання (від 0 до *fV*) з майже плоскою характеристикою (нерівномірність 0.001 дБ).

2. Зону переходу (від *fV* до *fs*−*fV*) з крутим спадом.

3. Смугу придушення (вище *fs*−*fV*) з рівнем придушення -75 дБ. Покажіть, що фільтр має 16-бітні коефіцієнти.

Нерівномірність коефіцієнта передачі такого фільтра у смузі пропускання становить лише 0.001 дБ. Оскільки КІХ-фільтри також мають лінійну фазову характеристику, вони практично не вносять фазових спотворень у вихідний сигнал.

Коли сигнал проходить через такий розроблений фільтр, його форма значно покращується (рис 3.14).

Рисунок 3.14 - Вихідний сигнал інтерполяційного фільтра

З цього графіку видно, що форма сигналу на виході фільтра дуже близька до оригінального аналогового сигналу (сигналу до дискретизації). Важливо зазначити, що сигнал на виході фільтра буде затриманий на час, що називається груповою затримкою фільтра. Для КІХ-фільтрів цей час приблизно дорівнює половині кількості відводів, помноженій на період тактування.

Оскільки АЧХ реального фільтра завжди має скінченний нахил, смуга частот інтерпольованого сигналу завжди повинна бути менше половини нової частоти дискретизації (*fs*/2).

Уникнення спотворень: Приклад з дзеркальними відображеннями Давайте подивимося, що станеться, якщо ми не зможемо повністю придушити перше дзеркальне відображення спектра (рис 3.15).

Рисунок 3.15 - Вихідний сигнал з присутністю дзеркального відображення

Цей графік чітко показує, що навіть невелике "просочування" високочастотних компонентів може значно спотворити вихідний сигнал. Присутність таких дзеркальних образів у спектрі є небажаною.

Багатостадійна інтерполяція

Інтерполяція часто виконується в кілька етапів. Перші два етапи зазвичай подвоюють частоту дискретизації сигналу. Це необхідно, тому що на початковому етапі корисний сигнал займає майже всю смугу частот від 0 до *fs*/2, що означає, що корисний сигнал та його перше високочастотне дзеркальне відображення знаходяться дуже близько. Тому вимоги до крутизни АЧХ інтерполяційного фільтра є дуже

високими, і для досягнення необхідної інтерполяції потрібна велика кількість відводів та складні коефіцієнти.

Після перших двох етапів інтерполяції корисний сигнал займає лише 25% нової, розширеної смуги частот. Це значно знижує вимоги до селективності фільтра, дозволяючи наступним інтерполяційним фільтрам забезпечувати вищий коефіцієнт інтерполяції з меншими обчислювальними затратами.

Важливо наголосити: проміжні відліки слід заповнювати саме нулями. Якщо їх заповнювати іншими значеннями (наприклад, повторюючи попередній відлік), рівень високочастотних складових у спектрі може бути знижений, але основний сигнал також буде спотворений. Крім того, в такому випадку фільтр, розроблений за принципами Найквіста, не працюватиме як коректний інтерполяційний фільтр, оскільки вихідні значення відліків сигналу будуть спотворені з самого початку.

РОЗДІЛ 4
ІНТЕРФЕЙС ТА ВІЗУАЛІЗАЦІЯ ДАНИХ

4.1 *USB*-інтерфейс для з'єднання з мікроконтролером

Універсальна послідовна шина (*USB*) є де-факто стандартом для підключення периферійних пристроїв до комп'ютерів, і *Arduino* активно використовує її для передачі даних. Символ *USB,* що складається з чотирьох геометричних фігур (квадрат, трикутник, велике та мале коло), візуалізує універсальність та розгалужену структуру підключень.

USB починався як інтерфейс для периферійних пристроїв з низькою та середньою швидкістю. Хоча раніше *FireWire* (*IEEE* 1394) вважався кращим для високошвидкісних пристроїв, вихід *USB* 3.0 (а згодом *USB* 3.1, *USB* 3.2 та *USB*4) повністю змінив цю ситуацію. Сучасні стандарти *USB* забезпечують високу пропускну здатність, яка значно перевищує потреби більшості периферійних пристроїв.

Кабель *USB* зазвичай складається з двох витих пар:

- Одна пара (*D*+ і *D*-) призначена для двонаправленої передачі даних за допомогою диференціальної сигналізації, що забезпечує стійкість до перешкод.
- Інша пара використовується для живлення підключених пристроїв (+5 В).

Початковий стандарт *USB* 2.0 забезпечував до 500 мА струму, що дозволяло живити багато малопотужних пристроїв без зовнішніх адаптерів або заряджати акумулятори портативних гаджетів (камер, плеєрів). Сучасні стандарти, такі як *USB* 3.0, збільшили доступний струм до 900 мА, а з появою *USB Power Delivery* (*USB PD*)

через роз'єми *USB-C* стало можливим передавати до 100 Вт потужності, дозволяючи живити та заряджати навіть ноутбуки.

З'єднання *USB*-кабелем створює інтерфейс між *USB*-пристроєм (наприклад, *Arduino*) та *USB*-хостом (комп'ютером). Хост використовує *USB*-контролер, який керується операційною системою і містить *USB*-хаб. Цей вбудований хаб, відомий як кореневий хаб (*root hub*), є відправною точкою для створення ланцюжка пристроїв,

що відповідає топології "зірка". До його портів можуть підключатися інші *USB* пристрої або зовнішні хаби. Загальна кількість пристроїв, що можуть бути підключені до одного кореневого хабу, не повинна перевищувати 127, а максимальна глибина каскадування хабів (без врахування кореневого) становить 5 рівнів.

USB-роз'єми спеціально розроблені для забезпечення "гарячого" підключення та відключення пристроїв без необхідності вимкнення живлення хоста. Це досягається завдяки тому, що контакти заземлення (*GND*) є довшими, ніж сигнальні контакти. Таким чином, при підключенні спочатку встановлюється стабільний потенціал заземлення між корпусами пристроїв, захищаючи електроніку від електростатичного розряду та електричних пошкоджень перед замиканням сигнальних ліній.

4.2 Робота з послідовними портами в *Java*: Бібліотека *jSSC*

Для програмної взаємодії з послідовними (*COM*) портами з використанням *Java*, однією з популярних та ефективних бібліотек є *jSSC* (*Java Simple Serial Connector*). Вона була офіційно випущена у 2010 році під ліцензією *LGPL*, ставши відповіддю на відсутність адекватних та крос-платформних інструментів для роботи з послідовними портами в той час (існуючі рішення, такі як *javax.comm* чи *rxtx*, мали певні обмеження або проблеми сумісності).

Бібліотека *jSSC* розділена на кілька основних компонентів:

- *SerialPortList* (оновлення *SerialLocalInterface*): Клас, що надає доступ до методів для отримання списку доступних послідовних портів у системі. - *SerialPort*:

Основний клас, який безпосередньо представляє послідовний порт і дозволяє ним керувати (відкривати, закривати, налаштовувати, читати та писати дані).

- *SerialPortEventListener*: Інтерфейс, який необхідно реалізувати, якщо ви бажаєте обробляти події, що відбуваються на послідовному порту (наприклад, надходження нових даних).

Розглянемо невеликий приклад. Припустимо, у нас є пристрій (наприклад, *Arduino*), що використовує апаратне керування потоком (*RTS*/*CTS*), і ми хочемо отримувати від нього відповіді.

```
import jssc.SerialPort;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;
import java.io.IOException; // Додаємо для потенційних
IOException public class ArduinoCommunicator {
 private static SerialPort serialPort; // Змінено назву класу на більш описову
public static void main(String[] args) {
 // Передаємо назву порту в конструктор.
 // На Linux/macOS це може бути "/dev/tty.usbmodemXXXX" або "/dev/ttyUSB0",
 // на Windows - "COM1", "COM2" тощо.
 serialPort = new SerialPort("COM1");
 try {
 // Відкриваємо порт
 serialPort.openPort();
 // Встановлюємо параметри послідовного зв'язку: швидкість, біти даних,  стоп-біти, парність
 serialPort.setParams(SerialPort.BAUDRATE_9600,
 SerialPort.DATABITS_8,
 SerialPort.STOPBITS_1,
 SerialPort.PARITY_NONE);
 // Вмикаємо апаратне керування потоком (RTS/CTS) для вхідних та вихідних
```

даних

serialPort.setFlowControlMode(*SerialPort.FLOWCONTROL_RTSCTS_IN* | *SerialPort.FLOWCONTROL_RTSCTS_OUT*);

53

// Налаштовуємо слухач подій та маску для отримання символів (коли доступні дані)

serialPort.addEventListener(*new DataReceiver*(), *SerialPort.MASK_RXCHAR*); // Змінено назву класу слухача

// Надсилаємо початковий запит до пристрою

serialPort.writeString("*GET_DATA\n*"); // Додано символ нового рядка для кращої сумісності

System.out.println("Запит '*GET_DATA*' надіслано.");

} *catch* (*SerialPortException ex*) {

System.err.println("Помилка послідовного порту: " + *ex.getMessage*());

ex.printStackTrace(); // Для детального виведення помилки }

}

// Внутрішній статичний клас для обробки подій послідовного порту *private static class DataReceiver implements SerialPortEventListener* { // Змінено назву класу

@Override

public void serialEvent(*SerialPortEvent event*) {

// Перевіряємо, чи подія пов'язана з прийомом даних (*RXCHAR*) і чи є доступні байти

if (*event.isRXCHAR*() && *event.getEventValue*() > 0) {

try {

// Зчитуємо доступні байти з буфера порту

byte[] *buffer = serialPort.readBytes*(*event.getEventValue*()); // Конвертуємо байти в рядок, використовуючи *UTF*-8 кодування *String receivedData = new String*(*buffer*, "*UTF*-8"); // Можна використовувати *StandardCharsets*.*UTF*_8;

System.out.println("Отримано дані: " + *receivedData*);

54

// Після обробки, якщо потрібно, можна надіслати новий запит або команду

```
// serialPort.writeString("ACK_DATA\n"); // Приклад підтвердження отримання
// System.out.println("Надіслано підтвердження.");
} catch (SerialPortException | IOException ex) { // Об'єднуємо винятки
System.err.println("Помилка при читанні/запису даних: " + ex.getMessage());
ex.printStackTrace();
}
}
}
}
}
```

Все виглядає досить просто та інтуїтивно зрозуміло. Окрім базових операцій читання/запису, *jSSC* надає широкий спектр можливостей:

- Керування лініями *RTS*, *DTR*: Дозволяє програмно встановлювати стани цих керуючих ліній.
- Отримання стану ліній *CTS*, *DSR*, *RING* та *RLSD*: Можливість моніторити стани цих сигнальних ліній.
- Отримання кількості байтів у вхідному/вихідному буфері: Дозволяє відстежувати обсяг даних, що очікують на обробку.
- Очищення буфера порту: Можливість скинути вміст вхідного або вихідного буфера.
- Генерація сигналу переривання (*Break signal*): Використовується для спеціальних цілей у послідовному зв'язку.
- Повне керування потоком: Підтримка апаратного (*RTS*/*CTS*) та програмного (*XON*/*XOFF*) керування потоком даних.
- Отримання списку *COM*-портів у системі: Дуже корисна функція для автоматичного виявлення доступних портів.

Важливо зазначити: *USB-to-COM* адаптери (які часто використовуються для підключення *Arduino* до комп'ютера) іноді можуть поводитися непередбачувано

порівняно з нативними *COM*-портами. Їхня поведінка значною мірою залежить від якості драйверів. Тому, якщо ви плануєте використовувати такі адаптери, ретельно тестуйте їх роботу в різних операційних системах (*Linux*, *Windows*, *macOS*), щоб переконатися в стабільній та коректній роботі, особливо якщо потрібна крос платформна підтримка.

За рік після випуску, застосування *jSSC* значно розширилося, включаючи такі проєкти:

- Системи управління *HTPC* (*Home Theater PC*)
- Модифікації автомобільної електроніки (наприклад, *Mitsubishi Eclipse*) -

Серверне програмне забезпечення для центрів мережевих технологій -

Системи зважування пакунків

- Численні освітні проєкти та розробки.

Ось приклад використання *jSSC* для запису даних ЕКГ у файл:

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.nio.charset.StandardCharsets; // Сучасний спосіб використання кодування

import java.util.ArrayList;

import java.util.List;

import jssc.SerialPort;

import jssc.SerialPortEvent;

import jssc.SerialPortEventListener;

import jssc.SerialPortException;

public class ECGDataLogger {

private static SerialPort serialPort;

// Використовуємо більш універсальний шлях для сучасних ОС або відносний шлях

private static final String FILE_PATH = "*ecg_data.txt*";

```
public static List<String> values = new ArrayList<>(); // Може бути використаний для тимчасового зберігання
public static void main(String[] args) {
// Шлях до порту Arduino на macOS/Linux може бути "/dev/tty.usbmodemXXXX"
// На Windows це буде "COMX" (наприклад, "COM3")
serialPort = new SerialPort("/dev/tty.usbmodem1411"); // Замініть на ваш порт
try {
serialPort.openPort();
serialPort.setParams(SerialPort.BAUDRATE_9600,
SerialPort.DATABITS_8,
SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);
// Встановлюємо маску подій, щоб слухати лише надходження даних
serialPort.setEventsMask(SerialPort.MASK_RXCHAR);
serialPort.addEventListener(new ComPortListener());
System.out.println("Порт відкрито та слухач налаштовано."); }
catch (SerialPortException ex) {
System.err.println("Помилка ініціалізації послідовного порту: " + ex.getMessage());
ex.printStackTrace();
}
}
private static class ComPortListener implements SerialPortEventListener {
@Override
public void serialEvent(SerialPortEvent event) {
if (event.isRXCHAR()) { // Якщо прийшли нові символи
try {
// Зчитуємо всі доступні байти з буфера порту
```



```
byte[] buffer = serialPort.readBytes(event.getEventValue()); if (buffer.length > 0) {
```

```
// Конвертуємо байти в рядок, використовуючи UTF-8 кодування String receivedMsg = new String(buffer, StandardCharsets.UTF_8);  // Записуємо дані у файл
try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_PATH, true))) {
writer.write(receivedMsg); // Записуємо отримані дані writer.newLine(); // Додаємо новий рядок, якщо дані приходять як окремі пакети
}
System.out.println("Отримано і записано: " + receivedMsg.trim()); // trim() для видалення зайвих пробілів
}
} catch (SerialPortException | IOException ex) {
System.err.println("Помилка читання/запису даних: " + ex.getMessage());
ex.printStackTrace();
}
}
}
}
}
```

4.3 Візуалізація ЕКГ-даних

Після збору даних через *USB*-інтерфейс, для їх візуалізації можна використовувати різні інструменти. Раніше часто використовувалися офісні програми.

Приклад візуалізації ЕКГ-сигналу, отриманого та обробленого системою можна побачити на рисунку 4.1.

Рисунок 4.1 - Приклад графіка ЕКГ

4.4 Алгоритм вимірювання частоти пульсу за отриманою ЕКГ

Для розрахунку частоти серцевих скорочень (пульсу) за отриманими ЕКГ даними ми застосуємо простий алгоритм, що базується на виявленні піків. Дані, що надходять з мікроконтролера, представлені значеннями від 0 до 1023 (для 10-бітного АЦП *Arduino*). Враховуючи особливості аналогового підсилювача, розробленого для ЕКГ (зміщення 2.5 В), очікувані значення сигналу знаходитимуться приблизно в діапазоні від 250 до 750.

Алгоритм розрахунку пульсу:

1. Виявлення *R*-зубців (піків): Для коректного розрахунку пульсу необхідно виділити найвищі піки серцевої активності – *R*-зубці *QRS*-комплексу. Це можна зробити, шукаючи максимальні значення в певному діапазоні, які перевищують поріг (наприклад, 80% від максимально можливого значення або динамічний поріг).

2. Вимірювання інтервалу *R*-*R*: Після виявлення *R*-зубців обчислюється час (або кількість відліків) між послідовними *R*-зубцями (інтервал *R*-*R*).

3. Перетворення в удари на хвилину:

- Ми знаємо, що буфер забезпечує 1000 відліків за секунду, тобто кожен відлік відповідає 1 мілісекунді (мс).

- Припустимо, ми вимірюємо інтервал *R*-*R* у відліках. Якщо інтервал *R*-*R* становить *N* відліків, то час між ударами становить *N* мс=*N*/1000 секунд. - Частота пульсу (у ударах на хвилину, *bpm*) розраховується за формулою:

(4.1) Де N – кількість відліків між двома *R*-зубцями.

Зазвичай, людське серце б'ється щонайменше 60 разів на хвилину (1 удар на секунду або частіше). Щоб виявити піки та розрахувати пульс, можна аналізувати дані протягом 2 секунд (тобто 2000 відліків). Система може оновлювати розрахунок пульсу кожні 2 секунди, надаючи актуальну інформацію.

Цей базовий алгоритм можна вдосконалювати за допомогою більш складних методів ЦОС (наприклад, адаптивних порогів, вейвлет-перетворення) для більш точного виявлення *R*-зубців та фільтрації артефактів руху або м'язових шумів, що підвищить надійність вимірювання пульсу.

60

ВИСНОВКИ

За бурхливим розвитком технологій не можливо встигнути. Ринок наповнюється усілякими рішеннями як для домашнього використання так і для професійного. Але медичних приладів на ринку дуже мало, особливо для домашнього використання. А їх захмарна ціна робить неможливим придбати їх для домашнього використання. Прості медичні прилади вже почали з’являтися в домах пересічних громадян, але не такі як ЕКГ. Платформа *Arduino*, для якої розроблявся модуль є доступною за ціною та достатньо потужною, щоб забезпечити будь-які забаганки користувача. Встановлення модулю та загрузка скетчу – тривіальні дії, які здатен зробити будь-який користувач ПК. Модуль ЕКГ, розроблений мною має ціну у 100 разів меншу ніж аналогічні продукти на ринку медичних приладів і саме це робить його унікальним. Цим проектом я хола показати, що навіть на найслабкішій платі *Arduino* можна зробити працюючий прототип такого складного пристрою як ЕКГ. І мені це вдалося. Розвиваючі такі технології, можна попередити серцево-судинні захворювання, від яких найчастіше помирають люди. На основі *Arduino* можна виготовити цілу біомедичну станцію, яка буде в автоматичному режимі діагностувати ти попереджати тяжкі захворювання. Маю надію, що завдяки моєму пристрою вдасться хоч трохи поліпшити рівень медицини в Україні та світі.

61

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Куксенко С. П. Мікроконтролери родини *PIC*: теорія та практика. – Львів : Видавництво Львівської політехніки, 2018. – 412 с.

1. Швець В. О. Основи мікроконтролерної техніки та програмування *Arduino*. – Київ : Центр учбової літератури, 2021. – 320 с.

2. Лавриненко Ю. М. Програмування мікроконтролерів *Arduino*: практичний курс. – Харків : Ранок, 2020. – 288 с.

3. Куксенко С. П. Основи електроніки та схемотехніки. – Львів : Видавництво Львівської політехніки, 2018. – 412 с. (Може бути актуальним для розуміння базових електронних компонентів).

4. Білоусов В. А. Мікропроцесорні системи та інтерфейси. – Київ : Видавничий дім "Києво-Могилянська академія", 2019. – 384 с. 5. Григорович В. В. Проектування вбудованих систем для біомедичних застосувань. – Дніпро : ДНУ, 2021. – 280 с.

6. Давидов С. О. Біомедична електроніка: сенсори та системи збору даних. – Вінниця : ВНТУ, 2016. – 180 с.

7. Іванов П. М. Розробка портативної системи моніторингу серцебиття на платформі *Arduino*. – *Вісник технічного університету*. – 2023. – № 1. – С. 65–72.
8. Коваленко Р. В. Алгоритми фільтрації та обробки ЕКГ-сигналів для вбудованих систем. – *Наукові записки НаУКМА. Серія: Комп'ютерні науки*. – 2024. – № 2. – С. 101–108.

9. Мельник Л. В. Використання оптичних сенсорів для вимірювання частоти серцевих скорочень з *Arduino*. – *Системи обробки інформації*. – 2022. – № 3. – С. 115–122.

10. Петренко Д. С. Бездротова передача даних у системах моніторингу здоров'я на базі *Arduino*. – *Комп'ютерні системи та мережі*. – 2023. – № 4. – С. 90–97.

11. *Arduino*. Офіційна документація та бібліотеки для платформ *Arduino*. – [Електронний ресурс]. – Режим доступу: https://docs.arduino.cc/ (дата звернення: 07.06.2025).

12. *IEEE Std* 11073-10101-2004. *Health Informatics — Point-of-Care Medical*

Device Communication — Part 10101: *Nomenclature — Approved Guide*. – Нью-Йорк : *IEEE*, 2004. – 80 с. (Загальний стандарт для медичних пристроїв, якщо релевантно).

13. *SparkFun Electronics*. Туторіали та проекти з *Arduino* та сенсорами. – [Електронний ресурс]. – Режим доступу: *https://learn.sparkfun.com/* (дата звернення: 07.06.2025).

14. *Adafruit Learning System*. Проекти та керівництва з електроніки та *Arduino*. – [Електронний ресурс]. – Режим доступу: *https://learn.adafruit.com/* (дата звернення: 07.06.2025).

15. *Instructables*. Проекти "зроби сам" з електроніки та *Arduino*. – [Електронний ресурс]. – Режим доступу: *https://www.instructables.com/* (дата звернення: 07.06.2025).

16. *Arduino Official Forum*. Форум спільноти для вирішення проблем та обговорення проектів. – [Електронний ресурс]. – Режим доступу: *https://forum.arduino.cc/* (дата звернення: 07.06.2025).

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу

випускника спеціальності: 123 «Комп'ютерна інженерія»

відділення: комп'ютерної та програмної інженерії

циклова комісія: комп'ютерних систем та мереж

Максим СОЛТАНОВ
(ім'я, прізвище)

1. Актуальність теми: Обрана тема кваліфікаційної роботи «Система моніторингу серцебиття з використанням платформи Arduino» є актуальною.

2. Кваліфікаційна робота відповідає темі, затвердженій наказом.

3. Завдання на виконання кваліфікаційної роботи виконано у повному обсязі.

4. В результаті виконання кваліфікаційної роботи була досліджена задача роботи датчику серцебиття на платформі Arduino.

5. Якість виконання пояснювальної записки та ілюстративного (графічного) матеріалу відповідає вимогам Державних стандартів.

6. В кваліфікаційній роботі зроблений акцент на дані отримані на практиці («живі» експерименти).

7. Кваліфікаційна робота заслуговує оцінку «добре».

Рецензент _____ викладач, «спеціаліст вищої категорії» _____
(науковий ступінь, посада)

«____»_____ 2025 р. _____ Андрій КРАВЧАТИЙ
(підпис) (ім'я, прізвище)

З рецензією ознайомлений _____ Максим СОЛТАНОВ
(підпис) (ім'я, прізвище)