НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Приладобудівний факультет
Кафедра інформаційно-вимірювальних технологій

«На правах рукопису»
УДК 621.3

До захисту допущено:
Завідувач кафедри

_____ Володимир ЄРЕМЕНКО
«___»_____2024 р.

Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Інформаційні вимірювальні технології»

зі спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка»
на тему: «Система збору експериментальних даних на основі потокових технологій»

Виконав:
студент VI курсу, групи ПІ-21мп
Наумовський Артем Андрійович _____

Науковий керівник:
доцент, к.т.н., доцент
Богомазов Сергій Анатолійович _____

Консультант з розділу «Розробка СТАРТАП-проєктів»
завідувач кафедри економічної кібернетики,
професор, д.е.н., професор
Бояринова Катерина Олександрівна _____

Рецензент:
професор, д.т.н., професор
Черепанська Ірина Юріївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2024 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) Приладобудівний факультет
(повна назва)

Кафедра інформаційно-вимірювальних технологій
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою «Інформаційні вимірювальні технології»

Спеціальність 152 «Метрологія та інформаційно вимірювальна техніка»
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Володимир ЄРЕМЕНКО

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Наумовський Артем Андрійович
(прізвище, ім’я, по батькові)

1. Тема дисертації: Система збору експериментальних даних на основі потокових технологій

Науковий керівник дисертації Богомазов Сергій Анатолійович, к.т.н., доц.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації: 10.01.2024 р.

3. Об’єкт дослідження: Мережеві інформаційно-вимірювальні системи дистанційно збору і збереження експериментальних даних

4. Предмет дослідження (вихідні дані): Організація апаратно-програмного забезпечення інформаційно-вимірювальних систем з потоковою обробкою даних (вимірювана величина - прискорення; кількість осей-3; інтерфейс - I2C; комунікаційні мережеві протоколи - TCP/IP, HTTP)

5. Перелік питань, які потрібно розробити:

5.1. Огляд особливостей використання потокових програмних технологій в системах збору даних.

5.2 Розробка апаратно-програмного забезпечення рівня збору даних на базі платформи ESP-8266.

5.3. Розробка серверної частини системи.
5.4. Розробка та дослідження демонстраційного макету системи.
5.5. Розробка стартап-проекту.

6. Перелік графічного (ілюстративного) матеріалу:
Макет демонстраційної системи, презентація, доповідь.
7. Орієнтовний перелік публікацій:
Програмне забезпечення для потокових систем збору експериментальних даних

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Стартап-проект	Бояринова К.О., д.е.н., проф, завідувач кафедри економічної кібернетики КПІ ім. І.Сікорського		

9. Дата видачі завдання 15.09.2023 року

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Огляд існуючих рішень для систем збору даних на базі потокових програмних технологій	до 01.11.2023 р.	
2.	Розробка апаратного-програмного забезпечення рівня збору даних	до 11.11.2023 р.	
3.	Розробка серверного програмного забезпечення	до 25.12.2023 р.	
4.	Мекетування та дослідження інформаційно-вимірювальної системи	до 31.12.2023 р.	
5.	Розробка стартап-проекту	до 01.01.2024 р.	
6.	Оформлення пояснювальної записки та ілюстративного матеріалу.	до 08.01.2024 р.	
7.	Підготовка дисертаційної роботи, доповіді-презентації	08.01.2024 – 10.01.2024	

Студент _____ Артем НАУМОВСЬКИЙ
(підпис) (ім'я, прізвище)
Науковий керівник дисертації _____ Сергій БОГОМАЗОВ
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Магістерська дисертація на тему: «Система збору експериментальних даних на основі потокових технологій», 90 сторінки, 3 додатків, 9 джерел.

Об'єкт дослідження: Мережеві інформаційно-вимірювальні системи дистанційного збору і збереження експериментальних даних**.**

Предмет дослідження: Організація апаратно-програмного забезпечення інформаційно-вимірювальних систем з потоковою обробкою даних (вимірювана величина - прискорення; кількість осей-3; інтерфейс - I2C; комунікаційні мережеві протоколи - TCP/IP, HTTP) **.**

Мета роботи: Розробка типової системи дистанційного технічного моніторингу, використання потокових програмних технологій при побудові таких систем.

Методи дослідження та апаратура: Аналіз існуючих рішень, робота з технічною документацією. Платформа ESP-8266, датчик акселерометр MPU-6050, мови програмування Python та C, платформа Arduino, інтерфейс I2C, бібліотека Quix Streams.

Результати роботи та сучасність продукту: Розроблено систему збору та обробки експериментальних даних на основі потокових технологій на базі платформи ESP-8266 з використанням з використанням бібліотеки Quix та мови програмування Python. Проведено дослідження роботи датчиків та програмного забезпечення, у результаті отримано стабільну роботу всіх компонентів системи.

Рекомендації щодо використання результатів роботи: Результати проведених досліджень можна використовувати при побудові систем збору експериментальних даних на основі потокових технологій**.** МІКРОСЕРВІС, ПРИСКОРЕННЯ, ПОТОКИ ДАНИХ, МЕРЕЖЕВІ ПРОТОКОЛИ, СИНХРОННІСТЬ, PYTHON, ARDUINO, ESP-8266, MPU-6050, I2C.

ABSTRACT

Master's thesis: "A system for collecting experimental data based on streaming technologies", 90 pages, 3 appendices, 9 sources.

Object of research: Networked information and measurement systems for remote collection and storage of experimental data.

Subject of research: Organization of hardware and software of information and measurement systems with streaming data processing (measured value - acceleration; number of axes - 3; interface - I2C; communication network protocols - TCP/IP, HTTP/REST)

Objective: Development of a typical remote technical monitoring system, study of the effectiveness of using asynchronous software technologies in the construction of such systems.

Research methods and equipment: Study of existing solutions, work with technical documentation. ESP-8266 platform, MPU-6050 accelerometer sensor, Python and C programming languages, Arduino platform, I2C interface, MongoDB database, Quix Streams library.

Results and state of the art of the product: A system for collecting and processing experimental data based on streaming technologies based on the ESP-8266 platform using the Python programming language has been developed. A study of the operation of sensors and software was carried out, as a result of which stable operation of all system components was obtained.

Recommendations for using the results of the work: The results of the research can be used in the construction of systems for collecting experimental data based on streaming technologies.

MICROSERVICE, ACCELERATION, DATA STREAMS, NETWORK PROTOCOLS, SYNCHRONIZATION, PYTHON, ARDUINO, ESP-8266, MPU-6050, I2C.

ЗМІСТ

ВСТУП

Аналітика в реальному часі дозволяє користувачам отримувати потоки даних, дуже швидко обробляти їх і отримувати потрібну інформацію або виконувати операції з даними реального часу або майже реального часу. Вона ґрунтується на технології потокової обробки подій, яка має дуже високу пропускну здатність.

Існує два типи аналітики у реальному часі. On-demand analytics (аналітика на запит) — надання даних або результатів обчислень користувачам або додаткам у режимі реального часу. Наприклад, надання показань телеметричних датчиків на відповідному сайті за запитом клієнта при випробуваннях автомобільної техніки.

Continuous analytics (безперервна аналітика) – обробка подій на постійній основі та потокова передача результатів кінцевим користувачам, додаткам або сховищам з даними.

Аналітика у реальному часі має багато застосувань у сучасному світі. Наприклад, вона може допомогти компаніям відстежувати і робити персоналізовані редагування, швидко та автоматично реагувати на зміни даних. Дуже поширене застосування полягає у обробці величезних обсягів даних журналів подій чи реакцій із IT-систем чи пристроїв інтернету речей (IoT) та використання їх для прийняття рішень. В роботі буде проведено аналіз вимог до потокової обробки даних, будуть розглянуті альтернативні рішення існуючих платформ та визначені шляхи їх покращення.

В роботі буде розроблено програмне забезпечення на мові програмування Python з використанням бібліотеки Quix для потокових систем збору експериментальних даних та розглянуто особливості його використання.

Використання даного програмного забезпечення дозволяє з мінімальними витратами ресурсів обробляти велику кількість інформації в режимі, що наближений до реального часу [1].

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТА РОБОТИ

1.1 Суть і роль потокової обробки даних

Інтернет речей (IoT) - це мережа фізичних пристроїв, транспортних засобів, приладів та інших фізичних об'єктів з вбудованими датчиками, програмним забезпеченням і мережевим підключенням для збору та обміну даними. Ці пристрої, які також називають "розумними об'єктами", варіюються від простих пристроїв для розумного будинку, таких як розумні термостати, до натільних пристроїв, таких як розумні годинники і одяг з підтримкою RFID, до складних промислових машин і транспортних систем. Інженери навіть уявляють собі "розумні міста", засновані на технологіях Інтернету речей.

Ефективна робота і повне використання всієї системи IoT залежить від мільйонів байтів інформації, що проходять через неї одночасно, і тому дуже важливо, щоб вона оброблялася правильно.

Правильна і своєчасна обробка великих обсягів даних є дуже важливим і складним процесом. Адже втрата або необробка даних часто може призвести до надзвичайних ситуацій.

1.2 Традиційна архітектура обробки даних

Традиційна архітектура програмного забезпечення для обробки даних в більшості випадків є орієнтованою на бази даних. Це означає, що для обробки даних необхідно неодноразово звернутися до бази даних (рис.1).

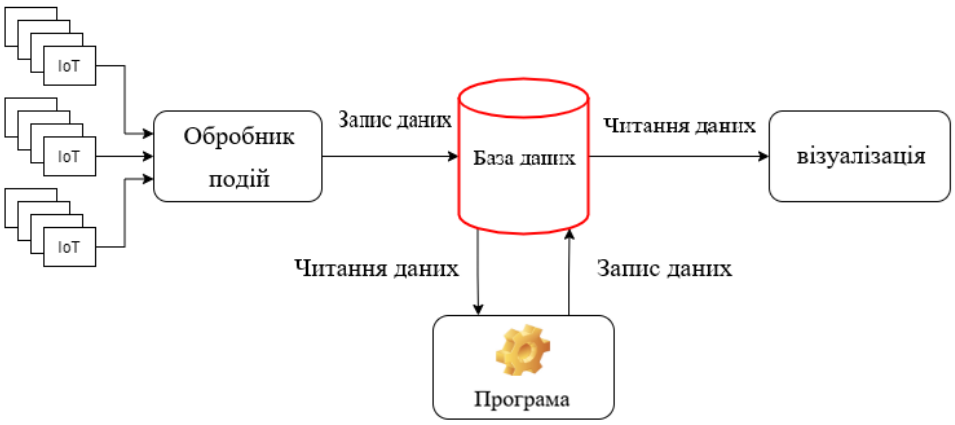


Рисунок 1.1 - Схема обробки даних на основі використання бази даних

Але в умовах обробки, що наближена до реального часу, потрібно отримувати результати в масштабі секунд або наносекунд. Наприклад, коли мільйони пристроїв IoT одночасно надсилають дані для обробки, традиційні архітектури, орієнтовані на бази даних, не задовольняють вимогам реального часу [2].

Потокова обробка використовується для отримання безперервного потоку даних та швидкої обробки подій в цьому потоці за невеликий проміжок часу після отримання даних. Час обробки може варіюватися від мілісекунд до хвилин. В багатьох випадках дані швидко деградують, тому за час, необхідний системі для збору, зберігання, обробки та надання даних, інформація часто змінюється. За допомогою штучного інтелекту, машинного навчання та обробки потоків у реальному часі можна використовувати дані способами, які інакше були б неможливими. Потокова обробка надає можливості реагувати в режимі реального часу і оперативно вносити потрібні зміни.

Для розробників які працюють з мовою програмування Python, реалізація програмного забезпечення на основі потокових технологій обробки даних є досить ускладненою. Це обумовлено тим, що ця мова програмування не сумісна з бібліотеками, що використовуються для реалізації потокових технологій (такими як Kafka), які переважно розроблені на інших мовах програмування, наприклад, Java та Scala. Платформа Kafka Streams була розроблена на мові програмування Java для забезпечення максимальної продуктивності. Мова програмування Python в ній підтримується для досить обмеженого кола задач, таких як публікація, підписка на повідомлення від брокера та виконання простих операцій [3]. Багато програмних платформ, які підтримують потокову обробку даних, не сумісні з мовою Python і розробникам доводиться переписувати код на інших мовах програмування.

Розглянемо принцип роботи потокової обробки даних.

Збір даних. Цей процес починається зі збору даних з джерел. Це можуть бути датчики, додатки, веб-сервіси або інші джерела, які генерують потоки

даних. Важливо забезпечити надійний та ефективний збір даних для подальшої обробки.

Збір і перетворення. Коли дані надходять, їх збирають і трансформують, щоб полегшити їхню обробку. Це включає фільтрацію, перетворення формату, агрегування та інші операції, необхідні для отримання інформації в потрібному форматі.

Обробка. Потім дані надсилаються на обробку. Тут застосовуються алгоритми і логіка для аналізу даних і виявлення закономірностей, аналізу тенденцій і аномалій. Це може бути виявлення атак на мережу, моніторинг стану обладнання, аналіз поведінки користувачів тощо.

Прийняття рішень. Після обробки даних система може приймати рішення на основі отриманих результатів. Це може бути автоматизована дія, наприклад, автоматичне виправлення виробничого процесу при виявленні невідповідностей, або ж надання інформації для прийняття рішень людиною, як у випадку аналізу фінансових даних.

Дія та зберігання. Залежно від результатів процесу, система може виконувати додаткові дії. Це може бути надсилання повідомлень, запис даних до сховища, архівування інформації, зміна робочих процесів тощо.

Розглянемо практичний сценарій. Уявімо систему моніторингу дорожнього руху в мерії міста. Датчики, прикріплені до транспортних засобів, безперервно передають інформацію про транспортний потік, швидкість, місцезнаходження та стан доріг. Ці дані негайно передаються в систему обробки потоку. На етапі обробки система аналізує дані, виявляє затори, визначає зони інтенсивного руху і виявляє аномалії в русі. На основі цих даних система автоматично розраховує оптимальний маршрут, попереджає про аварійні ситуації та радить водіям об'їжджати затори. Таким чином, потік обробки даних у цьому сценарії дозволяє адміністрації міста миттєво реагувати на дорожні умови, оптимізувати рух і підвищити безпеку дорожнього руху.

Розглянемо переваги та недоліки в порівнянні з пакетною обробкою.

Наведемо переваги обробки потокових даних. Реальний час. Однією з найважливіших переваг потокової обробки є можливість аналізувати дані в режимі реального часу. Це особливо важливо в сценаріях, де потрібна негайна реакція на подію, наприклад, фінансові транзакції або моніторинг обладнання.

Актуальність. Потокова обробка дозволяє отримувати актуальну інформацію про стан системи та події, що відбуваються навколо. Це важливо для прийняття швидких рішень і реагування на зміни в режимі реального часу.

Ефективність використання ресурсів. Обробка даних в міру їх надходження позбавляє від необхідності зберігати в пам'яті великі обсяги даних і ресурсоємні процеси. Це сприяє більш ефективному використанню обчислювальних ресурсів.

Зменшення затримок. Аналіз вхідних даних зменшує затримки при реагуванні на інциденти. Це особливо важливо для систем, де найменша затримка може мати фатальні наслідки.

Далі розглянемо недоліки потокової обробки даних.

Складність обробки. Обробка даних у реальному часі може бути складним завданням і вимагає оптимізації алгоритмів і обчислювальних процесів для досягнення прийнятної продуктивності.

Управління затримками. У деяких сценаріях затримка між надходженням даних і їх обробкою може бути неприйнятно великою. Управління такими затримками вимагає спеціальних методів.

Складність налагодження. Налагодження та тестування в потокових системах може бути складнішим, ніж в системах пакетної обробки, через динамічну природу даних та асинхронну обробку.

Архітектурні компоненти обробки потокових даних.

Джерело даних. У контексті потокової обробки під джерелами даних маються на увазі ресурси, з яких безперервно надходить інформація для подальшої обробки. Розглянемо, що включають в себе ці джерела.

Датчики та пристрої IoT. У світі Інтернету речей (IoT) датчики в пристроях безперервно збирають дані про стан навколишнього середовища,

такі як температура, вологість, рух та інші параметри. Ці дані відіграють важливу роль у моніторингу та управлінні процесами.

Журнали додатків. Веб-додатки, сервери та інші системні програми і сервіси генерують журнали, що містять цінну інформацію про системні процеси, помилки, події та запити. Аналіз логів у режимі реального часу може допомогти виявити проблеми та аномалії.

Соціальні мережі. Соціальні платформи генерують великі обсяги даних про поведінку, інтереси та взаємодію користувачів. Аналізуючи потоки даних із соціальних мереж, компанії можуть зрозуміти думку користувачів і адаптувати свої маркетингові стратегії.

Системи моніторингу та управління. В інфраструктурному та промисловому секторах багато систем моніторингу та управління генерують дані про стан обладнання, виробничі процеси та споживання енергії. Ці дані важливі для забезпечення ефективної роботи та запобігання нещасним випадкам.

Гарантована доставка та управління затримками. Одним з ключових аспектів роботи з джерелами даних в потоковій обробці є гарантована доставка та управління затримками. Потокова обробка вимагає, щоб дані були точними та актуальними, тому важливо, щоб вони надходили до процесора негайно і без втрат. У цьому розділі пояснюється, що включають в себе принципи гарантованої доставки.

Відмовостійкість. Системи потокової обробки повинні бути спроектовані з урахуванням можливих збоїв і перерв у передачі даних. Для гарантування надійної доставки можна використовувати механізми повторної передачі та виявлення помилок.

Управління затримками. Затримки в передачі та обробці даних можуть вплинути на достовірність аналізу. Тому важливо мати механізми для управління затримками та оптимізації часу доставки даних.

Буферизація і масштабованість. Буферизація може бути використана для обробки великих обсягів даних і забезпечення толерантності до перехідних

навантажень. Буферизація може стабілізувати перехідні піки і дозволяє системі ефективно працювати при високих навантаженнях. Важливо розуміти, що різні джерела даних вимагають гнучких і адаптивних рішень. Наприклад, якщо ви працюєте з даними з датчиків Інтернету речей, вам може знадобитися оптимізація для високих частот передачі даних, або якщо ви працюєте з соціальними мережами, вам може знадобитися обробка і фільтрація великих обсягів інформації. Оптимізація та налаштування джерел даних є ключовим фактором для успішного впровадження архітектури потокової передачі даних.

Охарактеризуємо які основні функції платформи потокової обробки.

Управління даними. Платформа надає інструменти для збору, передачі та обробки даних з різних джерел. Можна створювати правила для фільтрації, трансформації та агрегування даних у режимі реального часу.

Обробка даних. Одним із ключових компонентів потокової платформи є система обробки даних. Системи обробки даних містять алгоритми для агрегації, аналізу, класифікації та інших операцій над потоками даних.

Управління затримками. Багато платформ можуть керувати затримками обробки даних. Це важливо для дотримання вимог до аналітичної достовірності в різних сценаріях.

Масштабованість. Платформи забезпечують масштабованість для обробки великих обсягів даних. Вони автоматично адаптуються до мінливих робочих навантажень і забезпечують стабільну продуктивність.

Управління станом. Потокові процеси можуть потребувати врахування стану процесу. Платформа надає інструменти для управління даними та станом процесів.

Інтеграція. Платформи потокової обробки часто підтримують інтеграцію з іншими системами та сервісами. Це дозволяє інтегрувати потокову обробку в більш широкий спектр архітектурних рішень.

Розглянемо приклади популярних платформ.

Apache Kafka - це розподілена платформа для потокового передавання даних та обміну повідомленнями. Розроблена для високопродуктивного збору,

передачі та зберігання даних у режимі реального часу, Kafka є популярним вибором для обробки великих обсягів даних, пропонуючи масштабованість та відмовостійкість.

Apache Flink - розподілена платформа для обробки даних у реальному часі та пакетної обробки даних, Flink підтримує складні аналітичні операції та високу продуктивність. Вона також надає функції управління затримками та обробки даних з урахуванням стану.

Apache Storm - система обробки даних у реальному часі, призначена для аналізу потоків даних з низькою затримкою; Storm може створювати складні топології обробки та забезпечує надійність і відмовостійкість.

Amazon Kinesis - платформа для обробки потокових даних, керована Amazon Web Services (AWS); Kinesis надає інструменти для збору, аналізу та візуалізації даних в режимі реального часу.

Вибір відповідної платформи для обробки потоків залежить від вимог проекту, масштабу завдання та екосистеми інструментів, що використовуються. Наприклад, Apache Kafka може бути найкращим вибором, якщо вам потрібно обробляти великі обсяги даних, тоді як Apache Flink може бути правильним рішенням, якщо вам потрібно виконати складний аналіз. Кожна платформа має свої переваги та обмеження, і важливо обрати ту, яка найкраще відповідає потребам конкретного проекту.

Етапи обробки та перетворення даних. Ці операції є центральною частиною потокового процесу. Він включає низку операцій з аналізу даних, вилучення важливої інформації та адаптації її для подальшого використання. Найпоширенішими операціями над потоками даних є фільтрація, трансформація та об'єднання. Фільтрація дозволяє відбирати дані на основі певних критеріїв. Наприклад, потік подій можна відфільтрувати, щоб вибрати лише ті, що відповідають певним критеріям. Перетворення даних дозволяє змінювати формат і структуру даних. Це включає зміну назв полів, перетворення типів даних та інші операції. Дані з різних джерел можна

об'єднати, щоб отримати повнішу картину. Наприклад, дані з різних датчиків можна об'єднати для більш точного аналізу подій.

При аналізі даних іноді необхідно враховувати часові інтервали, наприклад, збирати дані за певний період часу. Для цього використовуються віконні функції, які групують дані в певні часові вікна і застосовують до них процес агрегації.

Існують різні типи віконних функцій, зокрема часові вікна, лічильники та вікна сеансів. У першому випадку дані групуються за часовими інтервалами, такими як години, дні або тижні. Це дозволяє збирати дані протягом певного періоду часу. У випадку лічильників дані групуються відповідно до кількості подій. Наприклад, дані можна аналізувати для кожних 1000 подій. У вікні Сесії дані групуються за сесіями. Сесії можуть бути визначені часовим інтервалом неактивних пауз між подіями.

Забезпечення низької затримки та високої продуктивності. При обробці даних у реальному часі необхідно мінімізувати затримки та досягти високої продуктивності системи. Для цього використовуються різні методи та підходи, описані нижче.

Паралельні та розподілені. Паралельні обчислення та розподілені системи можна використовувати для ефективної обробки великих обсягів даних.

Компактне представлення даних. Оптимізація представлення даних може зменшити навантаження на мережу і пам'ять та зменшити затримки.

Використання кешу. Кешування результатів попередніх операцій може зменшити обсяг обчислень і прискорити обробку.

Управління пам'яттю. Ефективне використання пам'яті може зменшити накладні витрати і підвищити продуктивність.

Обробка та перетворення даних у потоковій обробці - це техніка, яка знаходить баланс між релевантністю, точністю та продуктивністю. Розробники повинні враховувати характеристики даних, архітектуру системи та вимоги кінцевого користувача. Ефективні методи потокової обробки даних можуть

призвести до створення більш оперативних, релевантних і значущих рішень для бізнесу та користувачів.

Зберігання даних у стані, проблеми зберігання даних у потоковій обробці. Зберігання стану в потоковій обробці є важливим і складним питанням. Стан - це інформація, яку система повинна зберігати між різними подіями, щоб гарантувати повноту і релевантність аналізу даних. Однак зберігання стану в потоковій обробці стикається з низкою проблем, серед яких

Масштабованість. При обробці великих обсягів даних сховище стану має ефективно масштабуватися.

То як забезпечити швидкий доступ до стану в умовах зростаючих навантажень і обсягів даних?

Управління станом. Як здійснюється управління станом в розподілених системах? Як забезпечити узгодженість даних та уникнути конфліктів при одночасному доступі?

Надійність. Стан повинен бути стійким до збоїв та помилок. Як забезпечується безпека даних у разі збоїв системи?

Латентність. Деякі операції над станами можуть займати багато часу, що може вплинути на загальну затримку в обробці даних. Як можна мінімізувати вплив затримок при роботі зі станами?

Розглянемо кілька підходів та рішень.

Зберігання в пам'яті. Сховища даних в пам'яті можуть значно прискорити доступ до стану. Цей тип сховища забезпечує швидкий доступ і зменшує затримки обробки завдяки зберіганню даних в оперативній пам'яті. Однак цей підхід вимагає управління пам'яттю і може бути обмежений обсягом пам'яті, доступної на пристрої.

Розподілені бази даних, такі як Cassandra, Apache HBase та Amazon DynamoDB, можна використовувати для забезпечення масштабованості та надійності зберігання даних. Такі бази даних можуть автоматично реплікувати дані для забезпечення відмовостійкості.

Системи кешування. Системи, такі як Redis та Memcached, можна використовувати для зберігання даних про стан, що часто використовуються. Зберігання в пам'яті забезпечує швидкий доступ до даних, але вимагає уваги до надійності та управління життєвим циклом даних.

Системи керування станами. Існують спеціалізовані системи керування станами, які надають механізми для зберігання, оновлення та запиту станів у потокових процесах. Ці системи можуть гарантувати узгодженість даних і відмовостійкість.

Полегшений патерн стану. Цей патерн означає, що стан не зберігається безпосередньо в потоковій системі, а переміщується в зовнішню систему. Потокова система зберігає лише стан, на який є посилання. Такий підхід допомагає зменшити навантаження на систему обробки та спростити управління станами.

Вдосконалені алгоритми та структури даних. Використання оптимізованих алгоритмів і структур даних може підвищити продуктивність обробки станів. Наприклад, фільтр Блума можна використовувати для швидкого визначення наявності елемента в наборі даних.

Використання зовнішніх сховищ даних для зберігання стану Багато потокових систем використовують зовнішні сховища даних для вирішення проблем зберігання стану. Це можуть бути розподілені бази даних, кеш-системи або інші сховища. Важливо вибрати відповідне сховище, виходячи з вимог проекту та характеристик даних. Важливою перевагою використання зовнішнього сховища даних є те, що зовнішнє сховище зазвичай забезпечує механізм масштабування, який може керувати збільшенням обсягів даних. Інші переваги включають узгодженість і надійність, відмовостійкість і різноманітність сховищ. Деякі сховища надають засоби для забезпечення узгодженості та надійності даних і можуть спростити управління станом. Зовнішні сховища даних можуть забезпечити збереження даних у разі збоїв системи, що важливо для підтримки цілісності держави. Існують різні типи сховищ даних, і ви можете вибрати той, який найкраще відповідає вашим

конкретним вимогам. До них відносяться реляційні бази даних, сховища NoSQL і кеші.

Сховище даних - один з ключових аспектів обробки потокових даних, який має безпосередній вплив на якість аналізу та швидкість реагування системи на зміни. Вибір правильного сховища даних і розробка ефективної стратегії управління станом вимагає глибокого розуміння вимог проекту і характеристик даних. Правильне рішення для зберігання даних може допомогти забезпечити надійність, продуктивність і релевантність аналізу потокових даних.

1.3 Мета роботи. Постановка вимог до системи

IoT дозволяє цим розумним пристроям взаємодіяти один з одним та з іншими пристроями з доступом до Інтернету, такими як смартфони та шлюзи, створюючи широку мережу взаємопов'язаних пристроїв, які можуть обмінюватися даними та виконувати різноманітні завдання автономно. Це може включати все: від моніторингу умов навколишнього середовища на фермах до управління схемами руху за допомогою розумних автомобілів та інших розумних автомобільних пристроїв, управління машинами та процесами на заводах, відстеження запасів і відвантажень на складах.

Потенційне застосування IoT широке та різноманітне, і його вплив вже відчувається в широкому спектрі галузей, включаючи виробництво, транспорт, охорону здоров'я та сільське господарство. Оскільки кількість пристроїв, підключених до Інтернету, продовжує зростати, IoT, ймовірно, відіграватиме все більш важливу роль у формуванні нашого світу та трансформації того, як ми живемо, працюємо та взаємодіємо один з одним.

У корпоративному контексті пристрої IoT використовуються для моніторингу широкого спектру параметрів, таких як температура, вологість, якість повітря, споживання енергії та продуктивність машини. Ці дані можуть бути проаналізовані в режимі реального часу, щоб виявити закономірності,

тенденції та аномалії, які можуть допомогти компаніям оптимізувати свою діяльність і підвищити прибуток.

Метою роботи є розробка програмного забезпечення на мові програмування Python з використанням бібліотеки Quix для потокових систем збору експериментальних даних. Використання даного програмного забезпечення дозволяє з мінімальними витратами ресурсів обробляти велику кількість інформації в режимі, що наближений до реального часу. Також будуть використовуватись датчики для отримання експериментальних даних та джерело цих даних у вигляді об’єкту, що буде постійно змінювати свій стан, генеруючи потоки інформації.

Вимоги до програмного забезпечення будуть наступними:

1. Бути масштабованим, відкритим для можливих змін для покращення взаємодії між компонентами або внесення інших змін відповідно до вимог.
2. Забезпечувати максимальну швидкодію між компонентами.
3. Бути спроектоване відповідно до всіх актуальних вимог до сучасного програмного забезпечення, враховуючи безпеку даних, стиль написання коду та використання актуальних та підтримуваних бібліотек, що дозволить легко адаптувати або покращити систему під різні технічні завдання.
4. Відмовостійкість. Програмне забезпечення має бути готовим до отримання некоректних даних чи до можливих збоїв.

2 РОЗРОБКА СТРУКТУРИ АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ЗБОРУ ДАНИХ

2.1 Проектування системи обробки потокових даних

Для розробки описаної системи обробки потокових даних ми будемо використовувати мову програмування Python, де буде використано бібліотеку Quix Streams, інтегровану базу даних MongoDB для зберігання отриманих даних та можливості їх подальшої аналітики, акселерометра MPU6050, що має фіксувати показники прискорення досліджуваного об’єкту, що і буде нашими потоковими даними, чіп ESP8266EX та об’єкт, що буде змінювати своє положення у просторі.

2.2 Python та Quix Streams

Використовувати переваги потокової обробки даних на основі технологій Python дозволяє бібліотека Quix Streams [4]. Це бібліотека з відкритим кодом для потокової передачі та обробки телеметричних даних. Вона призначена для використання в службах високочастотної телеметрії, коли необхідно обробляти великі обсяги даних часових рядів з точністю до наносекунд. Розроблена для Python клієнтська бібліотека Quix Python дозволяє фахівцям з обробки та аналізу даних обробляти потокові дані та розгортати моделі машинного навчання та інформаційні продукти без втручання розробників та інших технічних фахівців. Quix Streams використовує брокер повідомлень Apache Kafka замість бази даних, що дозволяє обробляти дані часових рядів з високою продуктивністю.

Бібліотека надає користувачу веб-інтерфейс, можливість використання онлайн IDE, зразки коду, готові до запуску та розгортання, швидке і просте розгортання потокової передачі та керування через інтерфейс користувача, інструменти моніторингу для відстеження стану і даних потокових програм у режимі реального часу, зручну візуалізацію у вигляді конвеєра та провідник даних для перевірки даних, оброблених програмою. Програмне забезпечення

має чотири види API (Application Programming Interface): Stream Writer API призначене для запису даних у систему обміну повідомленнями між серверними додатками Kafka за допомогою HTTP і містить методи шифрування, серіалізації та перетворення даних у формат Quix Streams незалежно від джерела даних; Stream Reader API відправляє дані у програму, що буде обробляти ці дані та забезпечує мінімальну затримку передачі даних, уникаючи будь-яких операцій з диском; Data Catalogue API містить записи попередніх потоків даних та дозволяє створювати інформаційні панелі та експортувати дані в інші системи; Portal API для автоматизації процесу передачі потоків даних та створення робочих областей.

Quix використовує брокер повідомлень і робить його центром програмного забезпечення, виконуючи обробку даних без необхідності зберігати та передавати інформацію через базу даних (рис. 2.1).

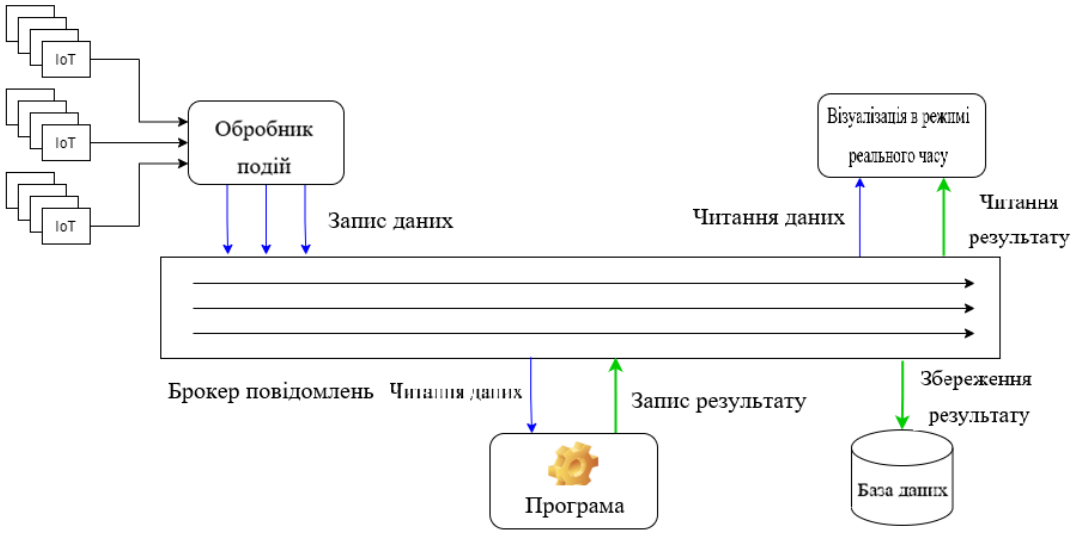


Рисунок 2.1 - Обробка даних за допомогою бібліотеки Quix

Використовуючи обробку в пам’яті програма може зберігати лише ті дані, які дійсно необхідно використати. Такий підхід знижує складність і вартість обробки даних в реальному часі на кілька порядків і дозволяє обробляти велику кількість даних при низьких затримках (рис. 2.2).

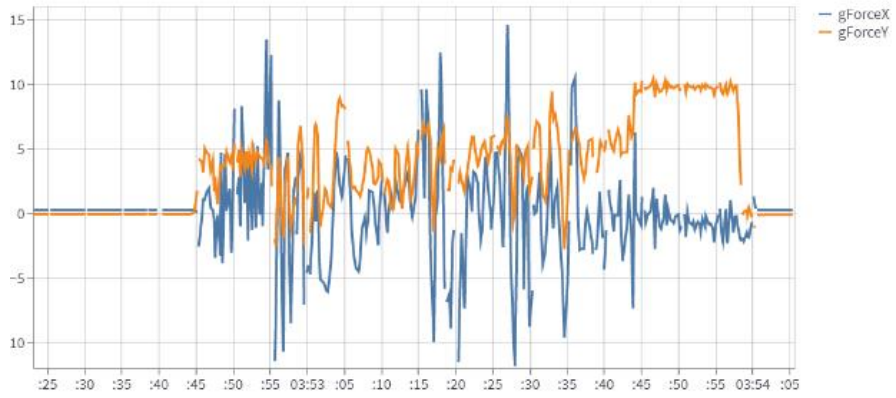


Рисунок 2.2 - Відображення даних від датчиків в режимі реального часу

Розглянемо опис та детальні характеристики модуль гіроскопа MPU6050 та чіп ESP8266EX (LoLin NodeMcu v3), які використовуються у даній роботі.

2.3 Використання датчика MPU6050 у системах збору потокових даних

Модуль MPU6050-це мікроелектромеханічна система(МЕМС) з вбудованим 3-осьовим акселерометром і 3-осьовим гіроскопом[5]. Датчики можна використовувати для вимірювання прискорення, швидкості, орієнтації, переміщення та багатьох інших параметрів, пов'язаних з рухом системи та об'єкта(рис.2.3).

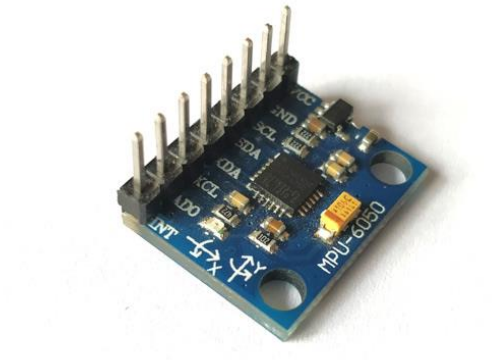


Рисунок 2.3- Модуль акселерометра та гіроскопа MPU6050

Модуль також має два допоміжні виводи, які можна використовувати для інтерфейсу з зовнішніми I2C модулями, такими як магнітометри, але це не є обов'язковим. Адреса I2C модуля налаштовується, тому до мікроконтролера

можна підключити декілька датчиків MPU6050 за допомогою виводу AD0. Модуль також постачається з добре задокументованою та перевіреною бібліотекою, що робить його дуже простим у використанні на відомих платформах, таких як Arduino. Тому датчик ідеально підходить для систем керування рухом, таких як радіокеровані автомобілі, дрони, самобалансуючі роботи та двоногі роботи (Таблиця 2.1).

Таблиця 2.1- Призначення виводів MPU6050

Пін-код	**Ім'я шпильки**	**Опис**
1	Vcc	Забезпечує живлення модуля, може бути від +3В до +5В. Зазвичай використовується +5 В
2	Землі	Підключений до заземлення системи
3	Серійний годинник (SCL)	Використовується для забезпечення тактового імпульсу для зв'язку I2C
4	Серійні дані (SDA)	Використовується для передачі даних через зв'язок I2C
5	Допоміжні послідовні дані (XDA)	Може використовуватися для інтерфейсу інших модулів I2C з MPU6050. Це необов'язково
6	Допоміжний послідовний годинник (XCL)	Може використовуватися для інтерфейсу інших модулів I2C з MPU6050. Це необов'язково
7	ДДУ	Якщо в одному MCU використовується більше одного MPU6050, то цей контакт можна використовувати для варіювання адреси
8	Переривання (INT)	Контакт переривання, щоб вказати, що дані доступні для читання MCU.

Апаратне забезпечення модуля дуже просте і фактично складається з MPU6050 як основного компонента, як показано на схемі вище. Оскільки модуль працює на 3,3 В, також використовується стабілізатор напруги. Лінія I2C підключена до живлення за допомогою резистора 4,7 кОм, а вивід AD0 заземлений за допомогою ще одного резистора 4,7 кОм (див. схему з'єднання на рис. 2.4).

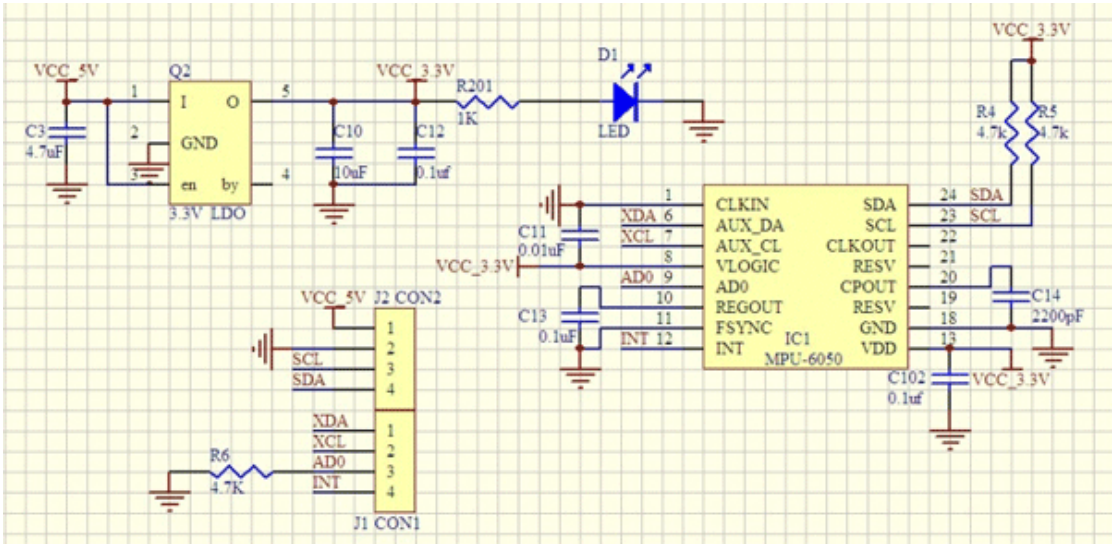


Рисунок 2.4 - Схема електрична принципова модуля датчика

Дані можна зчитувати з модуля MPU6050 через шину I2C. Зміни руху впливають на механічну систему, яка, в свою чергу, змінює напругу. Мікросхема має 16-бітний АЦП, який точно зчитує ці зміни напруги, зберігає їх у буфері FIFO і подає високий рівень на вивід INT (переривання). Це означає, що дані тепер готові до зчитування, тому MCU використовується для зчитування даних з цього буфера FIFO через зв'язок I2C. Існує багато платформ, таких як Arduino, де ви можете негайно почати використовувати цей модуль.

Розглянемо особливості гіроскопа. Трьохосьовий MEMS-гіроскоп у MPU-60X0 має широкий набір функцій:

- Датчики кутової швидкості X-, Y- та Z-осі з цифровим виходом (гіроскопи) із програмованим користувачем діапазоном повної шкали ±250, ±500, ±1000 та ±2000°/с
- Зовнішній сигнал синхронізації, підключений до контакту FSYNC, підтримує синхронізацію зображення, відео та GPS
- Вбудовані 16-розрядні АЦП дозволяють одночасну дискретизацію гіроскопів
- Покращена температурна стабільність зміщення та чутливості зменшує потребу в калібруванні користувачем

- Покращена продуктивність низькочастотного шуму
- Цифрово програмований фільтр низьких частот
- Робочий струм гіроскопа: 3,6 мА
- Струм у режимі очікування: 5 мкА
- Відкалібрований на заводі масштабний коефіцієнт чутливості
- Самоперевірка користувача

Розглянемо особливості акселерометра. Трьохосьовий MEMS-акселерометр у MPU-60X0 має широкий набір функцій:

- Трьохосьовий акселерометр із цифровим виходом і програмованим повним діапазоном шкали ±2g, ±4g, ±8g та ±16g
- Інтегровані 16-розрядні АЦП дозволяють проводити одночасну вибірку для акселерометрів
- Початкова похибка 3%
- Нелінійність 0,5%
- Нормальний робочий струм акселерометра: 500 мкА
- Струм у режимі акселерометра низької потужності: 10 мкА при 1,25 Гц, 20 мкА при 5 Гц, 60 мкА при 20 Гц, 110 мкА при 40 Гц
- Виявлення орієнтації та сигналізація
- Програмовані користувачем переривання
- Переривання High-G
- Самоперевірка користувача

MPU-60X0 складається з трьох незалежних MEMS-гіроскопів, які сприймають обертання по осях X, Y і Z. Коли гіроскоп обертається навколо однієї з осей датчика, ефект Коріоліса створює вібрації, які реєструються ємнісним датчиком. Отриманий сигнал підсилюється, демодулюється і фільтрується для отримання напруги, пропорційної кутовій швидкості. Ця напруга оцифровується за допомогою окремої інтегрованої мікросхеми у вигляді 16-розрядного аналого-цифрового перетворювача (АЦП), який здійснює вибірку по кожній осі. Повний діапазон гіроскопів можна

запрограмувати на ±250, ±500, ±1000 або ±2000 градусів за секунду (dps); АЦП можна запрограмувати від 8000 відліків за секунду до 3,9 відліків за секунду, а користувач може вибрати низькочастотний фільтр, який забезпечує широку частоту зрізу. Тривісні акселерометри MPU-60X0 використовують окремі тестові вантажі для кожної осі.

Прискорення вздовж певної осі викликає зміщення відповідного тестового вантажу, і ємнісний датчик визначає це зміщення по-різному. MPU-60X0 використовує архітектуру, яка знижує чутливість акселерометра до виробничих варіацій і теплового дрейфу Коли MPU-60X0 розміщений на плоскій поверхні, осі X і Y дорівнюють 0 g, а вісь Z - +1 g. Масштабний коефіцієнт акселерометра відкалібрований на заводі і номінально не залежить від напруги живлення. Кожен датчик має спеціальний сигма-дельта АЦП, який забезпечує цифровий вихід. Повний діапазон шкали цифрового виходу можна встановити в межах ±2g, ±4g, ±8g і ±16g.

MPU-60X0 взаємодіє з системним процесором через інтерфейс SPI (тільки MPU-6000) або послідовний інтерфейс I2C. MPU-60X0 завжди діє як ведений модуль при спілкуванні з системним процесором. Значення молодшого біта (LSB) адреси веденого модуля I2C встановлюється на виводі 9 (AD0):

- MPU-6000: Логічний рівень зв'язку з ведучим встановлюється напругою VDD
- MPU-6050: Логічний рівень зв'язку з ведучим встановлюється напругою VLOGIC.

MPU-60X0 має допоміжну шину I 2C для зв'язку з 3-осьовими цифровими магнітометрами та іншими датчиками. Ця шина має два режими роботи:

- Основний режим I 2C: MPU-60X0 виступає в ролі ведучого для зовнішніх датчиків, підключених до веденого пристрою шини I2C.
- Наскрізний режим: MPU-60X0 з'єднує провідну шину I2C безпосередньо з підлеглою шиною, дозволяючи системному процесору безпосередньо обмінюватися даними з зовнішнім датчиком.

Опишемо допоміжний режим роботи I2C. Ведучий режим I2C: MPU-60X0 може безпосередньо отримувати доступ до даних зовнішніх цифрових датчиків, таких як магнітометри. У цьому режимі MPU-60X0 отримує дані безпосередньо від допоміжного датчика, а вбудований DMP може генерувати дані об'єднання датчиків без втручання процесора системних додатків. Наприклад, в режимі ведучого I2C MPU-60X0 виконує пакетне зчитування і повертає наступні дані з магнітометра

- Дані магнітометра X (2 байти)
- Дані магнітометра Y (2 байти)
- Дані магнітометра Z (2 байти)

Майстер I2C може бути налаштований на зчитування до 24 байт з чотирьох допоміжних датчиків; п'ятий датчик може бути налаштований на роботу в режимі зчитування/запису 1 байта.

Наскрізний режим: зовнішній системний процесор виступає в ролі ведучого і безпосередньо взаємодіє з зовнішніми датчиками, підключеними до допоміжних виводів I2C (AUX_DA, AUX_CL). У цьому режимі логіка керування допоміжною шиною I2C (блок інтерфейсу зовнішнього датчика) MPU-60X0 відключена, а допоміжні виводи I2C AUX_DA і AUX_CL (виводи 6 і 7) через аналоговий перемикач підключаються до основної шини I2C (виводи 23 і 24). Наскрізний режим корисний для конфігурації зовнішніх датчиків або для підтримки MPU-60X0 в режимі низького енергоспоживання, коли використовуються тільки зовнішні датчики. У наскрізному режимі системний процесор все ще може отримати доступ до даних на MPU-60X0 через інтерфейс I2C. Допоміжна шина I2C має логічний рівень вводу/виводу, у MPU-6050: допоміжна шина I2C включає в себе логічний рівень VDD або VLOGIC програмований. На схемі нижче (рис. 2.5) системний процесор є ведучим I2C для MPU-60X0. MPU-60X0 також є ведучим I2C для додаткового зовнішнього датчика компаса; MPU-60X0 має обмежену функціональність як ведучий I2C і покладається на системний процесор для початкової конфігурації допоміжних датчиків. MPU-60X0 має мультиплексор обходу інтерфейсу, який

безпосередньо з'єднує виводи 23 і 24 шини I2C системного процесора (SDA і SCL) з виводами 6 і 7 шини I2C допоміжного датчика (AUX_DA і AUX_CL).

Після конфігурації допоміжного датчика системним процесором мультиплексор обходу інтерфейсу повинен бути вимкнений, щоб допоміжний майстер MPU-60X0 I2C міг керувати шиною датчика I2C і збирати дані з допоміжного датчика.

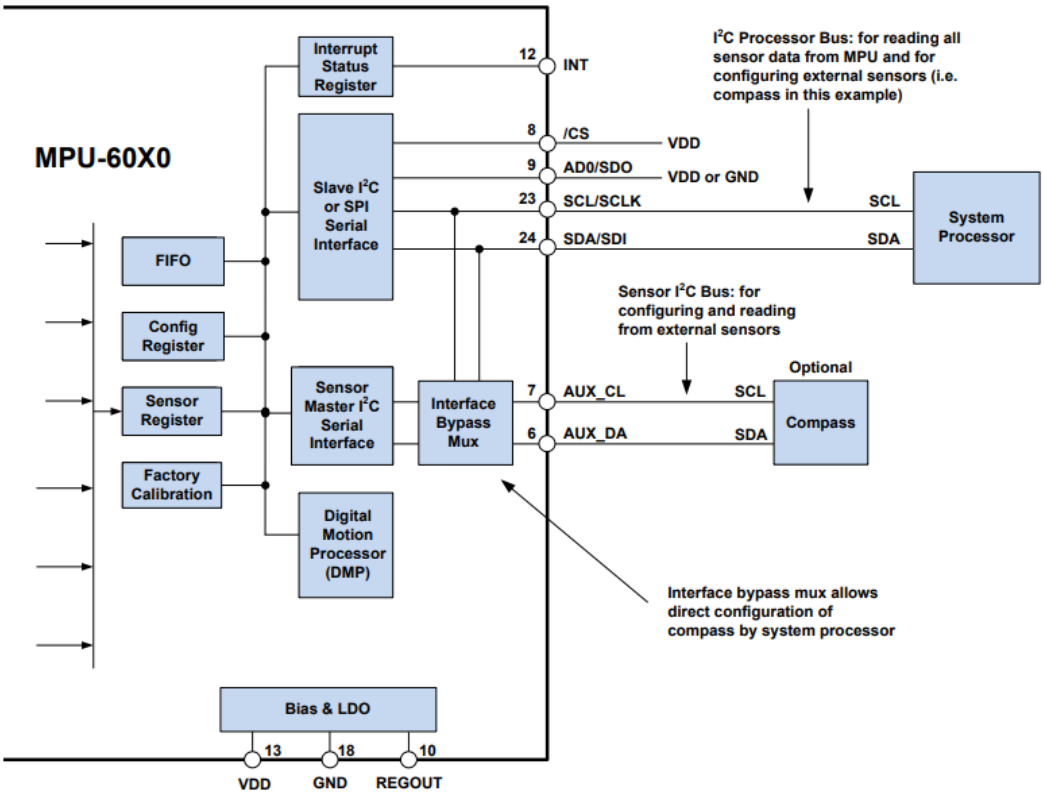


Рисунок 2.5 – Будова шини I2C у датчиках серії MPU-60X0

Регістри даних датчиків містять останні дані вимірювань гіроскопа, акселерометра, допоміжного датчика та температури. Ці регістри доступні лише для читання і доступні через послідовний інтерфейс. Дані в цих регістрах можуть бути прочитані в будь-який час. Однак функція переривання може бути використана, щоб визначити, коли доступні нові дані.

2.4 Використання модуля ESP8266EX у системах збору потокових даних

Чіп ESP8266EX від Espressif - це високоінтегроване рішення Wi-Fi SoC, яке відповідає вимогам індустрії IoT щодо низького енергоспоживання, невеликого розміру та надійності [6]. Мікросхему можна побачити на малюнку нижче (рис. 2.6).

Рисунок 2.6 - Модуль ESP8266EX

Завдяки повному стеку Wi-Fi та мережевих функцій, мікросхема ESP8266EX може запускати додатки самостійно або під управлінням зовнішнього мікроконтролера. При автономній роботі ESP8266EX завантажує і виконує програми з зовнішньої флеш-пам'яті. Вбудований високошвидкісний кеш підвищує продуктивність системи і дозволяє ефективно використовувати оперативну пам'ять. Працюючи під управлінням зовнішнього мікроконтролера, ESP8266EX може діяти як адаптер Wi-Fi і передавати дані через інтерфейси SPI, SDIO, I2C або UART.

ESP8266EX містить антенний перемикач, узгоджувальний трансформатор, підсилювач потужності, малошумящий підсилювач, фільтр і

модуль керування живленням. Компактна конструкція і високий ступінь інтеграції мінімізують площу друкованої плати і кількість зовнішніх компонентів.

ESP8266EX включає в себе вдосконалену версію 32-розрядного процесора Lx106 серії Diamond від Tensilica та SRAM на кристалі. Він може бути підключений до зовнішніх датчиків та інших пристроїв через інтерфейс вводу/виводу загального призначення (GPIO). Комплект розробки програмного забезпечення (SDK) містить приклади програм для різних застосувань.

Приклади застосування мікросхеми ESP8266EX включають побутову техніку, системи автоматизації, керовані розетки і лампи, mesh-мережі, промислові бездротові пристрої, радіоняні, веб-камери, сенсорні мережі, носимі пристрої. Wi-Fi позиціонування, бейджі, системи контролю доступу, системи Wi-Fi позиціонування тощо. Розглянемо структуру мікросхеми ESP8266EX та призначення портів вводу/виводу (рисунок 2.7 та таблиця 2.2).

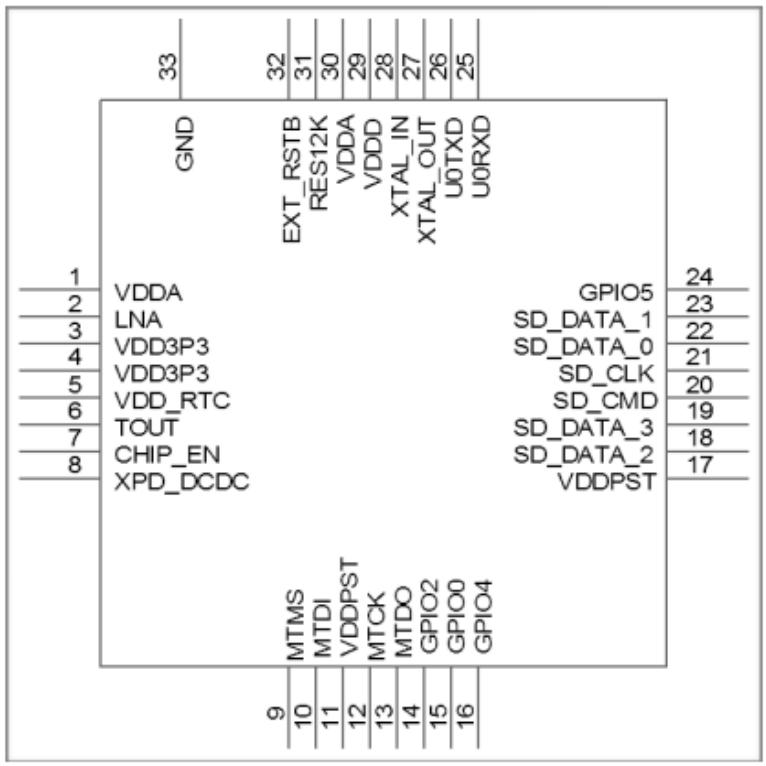


Рисунок 2.7 – Порти вводу-виводу ESP8266EX

Таблиця 2.2 – Основні технічні параметри ESP8266EX

Wi-Fi	Сертифікати	FCC/CE/TELEC/SRRC
	Протоколи Wi-Fi	802.11 b/g/n
	Діапазон частот	2,4 ГГц ~ 2,5 ГГц (2400 М ~ 2483,5 М)
	Потужність передачі сигналу	802.11 b: +20 дБм, 802.11 g: +17 дБм, 802.11 n: +14 дБм
	Чутливість приймача	802.11 b: -91 дом (11 Мбит/с) 802.11 g: -75 дБм (54 Мбит/с) 802.11 n: -72 дБм (MCS7)
	Типи антени	Антена на друкованій платі, зовнішня антена, керамічна чип-антена
Апаратне забезпечення	Центральний процесор	32-бітний мікропроцесор Tensilica Lx106
	Периферійні інтерфейси	UART/SDIO/SPI/I2C/I2S/IR Remote Control GPIO/ADC/PWM
	Робоча напруга	3.0В ~ 3.6В
	Робочий струм	Середнє значення 80 мА
	Діапазон робочих температур	-40°C ~ 125°C
	Робоча температура навколишнього середовища	Нормальна
	Розмір корпусу	5x5 мм
	Зовнішній інтерфейс	N/A
Програмне забезпечення	Режими Wi-Fi	кліент (station)/точка доступу (softAP)/SoftAP +station
	Безпека	WPA/WPA2
	Шифрування	WEP/TKIP/AES
	Оновлення ПЗ	Загрузка через UART / Дистанційне оновлення через мережу
	Розробка ПЗ	SDK для швидкої розробки ПЗ, інструменти для взаємодії з

		серверним ПЗ
	Мережеві протоколи	IPv4, TCP/UDP/HTTP/FTP
	Конфігурація користувача	Управління за допомогою AT команд, хмарний сервер, додаток для Android/iOS

Функціональна схема ESP8266EX показана на рисунку 2.8.

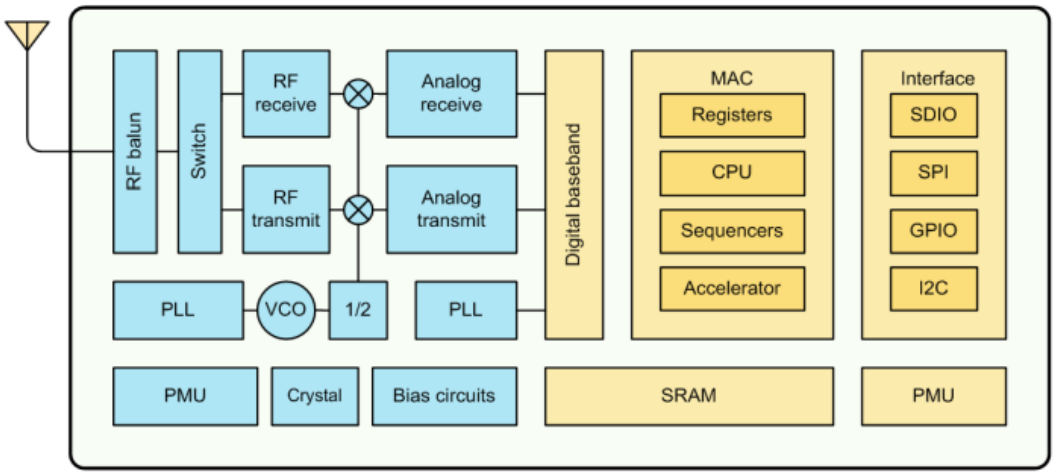


Рисунок 2.8 - Функціональна схема пристрою ESP8266EX

ESP8266EX містить 32-бітний мікропроцесор Tensilica lx106. Тактова частота процесора становить 80 МГц і може бути розігнана програмним забезпеченням до 160 МГц. Система може легко працювати як операційна система реального часу (RTOS). В даний час стек Wi-Fi використовує лише 20 відсотків процесорного часу. Час, що залишився, може бути використаний для користувацьких додатків. Мікропроцесор може взаємодіяти з іншими частинами мікросхеми через наступні інтерфейси: програмований інтерфейс RAM/ROM (iBus) з'єднується з контролером пам'яті і забезпечує доступ до зовнішньої флеш-пам'яті, інтерфейс пам'яті даних (dBus), який підключається до контролера пам'яті, інтерфейс AHB, що забезпечує доступ до периферійних регістрів. Wi-Fi SoC ESP8266EX містить контролер пам'яті і блоки пам'яті, включаючи SRAM (оперативна пам'ять) і ROM (постійна пам'ять), і блоки пам'яті ROM. Мікроконтролер може отримати доступ до блоків пам'яті через інтерфейси iBus, dBus і AHB.

У поточній версії SDK обсяг оперативної пам'яті, доступний користувачеві при роботі в режимі WiFi STA, становить 50 КБ. Цей обсяг оперативної пам'яті включає сегменти стеку, даних та купи. Оскільки ESP8266EX не має програмованого ПЗП, програма користувача зберігається у зовнішній SPI флеш-пам'яті. Сама ж зовнішня SPI флеш-пам'ять, підключена до ESP8266EX, використовується для зберігання програми користувача. Фактично підтримується до 16 МБ пам'яті. Рекомендується вибирати розмір флеш-пам'яті SPI наступним чином:

- Без OTA: мінімальний розмір флеш-пам'яті становить 512 КБ.
- З OTA мінімальний розмір флеш-пам'яті становить 1 МБ

Важливо звернути увагу, що підтримуються лише деякі з режимів SPI: Standard SPI, Dual SPI, DIO SPI, QIO SPI і Quad SPI. Необхідно вибрати правильний режим SPI під час завантаження двійкових файлів до флеш-пам'яті.

Крім арбітра, шина AHB включає в себе MAC, SDIO (хост) і процесор. Залежно від адреси, запит даних пересилається з AHB до одного з наступних двох підлеглих блоків: Шина даних ARB, контролер флеш-пам'яті (зазвичай для автономних додатків). В принципі, запити до контролера флеш-пам'яті є швидкими запитами, в той час як запити до блоку ARB є доступом до регістрів. Блок ARB діє як дешифратор для доступу до програмованих регістрів ESP8266EX. Залежно від адреси, APB запити можуть бути зроблені через радіо, I2S, SPI, SDIO (хост), GPIO, UART, годинник реального часу (RTC), MAC (RTC), або через модем.

ESP8266EX реалізує повний протокол 802.11 b/g/n/e/i WLAN MAC з підтримкою специфікації Wi-Fi Direct і дозволяє працювати в режимі BSS і в групах P2P відповідно до новітнього протоколу Wi-Fi P2P. Всі низькорівневі функції протоколу WiFi ESP8266EX виконує автоматично. Він спеціально розроблений для мобільних пристроїв, натільної електроніки та додатків IoT.

Має три режими енергоспоживання: активний режим, сплячий режим і режим глибокого сну. Енергоспоживання в режимі глибокого сну становить приблизно 20 мкА (при роботі з таймером реального часу) і менше 1,0 мА (при

роботі з таймером реального часу). Струм, споживаний для підтримки з'єднання з точкою доступу, становить менше 1,0 мА (DTIM=3) або 0,6 мА (DTIM=10). Підтримує з'єднання з точкою доступу. Коли вимкнено, то напруга низька на виході CHIP_PU. Таймер реального часу вимкнено, всі записи скидаються.

У режимі глибокого сну активним залишається лише таймер реального часу.

Коли вимкнено, то резервна пам'ять таймера реального часу може містити основну інформацію про з'єднання Wi-Fi. У сплячому режимі активний лише таймер реального часу, а кристалічний генератор вимкнений. Мікросхема прокидається, коли відбувається подія пробудження (MAC-адреса, хост, таймер реального часу або зовнішнє переривання. У стані пробудження система переходить зі сплячого режиму в режим увімкнення, де кристалічний генератор і ПЛІС вмикаються. Увімкнено- тоді високочастотний тактовий генератор увімкнено, і сигнали надсилаються до інших блоків відповідно до регістра керування тактовим генератором. Керування тактовою частотою також виконується в окремих блоках (рис.2.9).

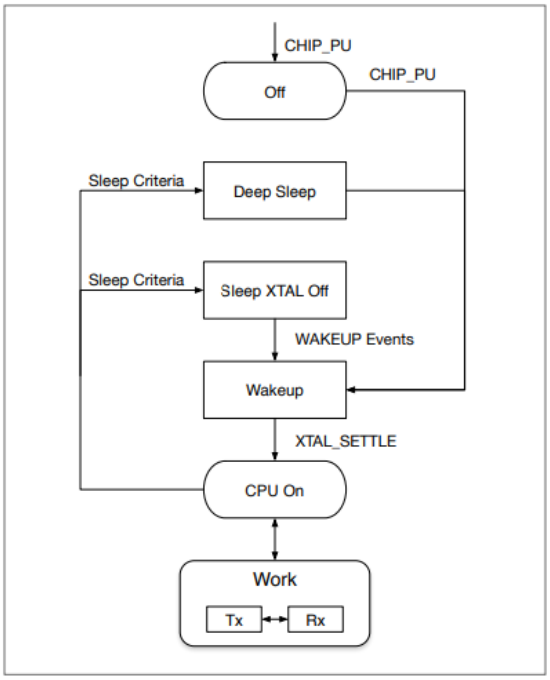


Рисунок 2.9 – Управління споживанням енергії

ESP8266EX має 17 виводів GPIO, які можна призначити для різних функцій, запрограмувавши відповідні регістри. Кожен GPIO є двонаправленим і неінвертуючим і внутрішньо підтягнутий до напруги живлення або землі.

При підключенні до напруги живлення або заземлення, він може бути переведений у високоомний стан. Високоімпедансний стан, це коли GPIO встановлено як вхід, вхідне значення можна зчитувати з регістра. Вхід також можна використовувати як джерело переривань залежно від рівня напруги або заземлення. Ці виводи можна комбінувати з іншими функціями, такими як I2C, I2S, UART, ШІМ та ІЧ. Для роботи в режимі енергозбереження виводи GPIO можуть підтримувати свій стан. Наприклад, всі виводи можуть бути відключені, коли мікросхема вимкнена. За необхідності в IO можуть бути інтегровані додаткові функції утримання. У наступних випадках, коли входи/виходи не контролюються внутрішніми або зовнішніми схемами, функція утримання може бути використана для збереження останнього використовуваного стану. Функція утримання забезпечує позитивний зворотний зв'язок з контактами. Тому зовнішній ланцюг, що керує контактами, повинен забезпечувати струм, більший за струм зворотного зв'язку. Необхідний струм невеликий, менше 5 мкА.

ESP8266EX має інтерфейс I2C, який використовується для підключення зовнішніх пристроїв, таких як мікроконтролери та датчики. У таблиці 2.3 описані виводи даного інтерфейсу.

Таблиця 2.3 – Описання виводів I2C інтерфейсу

Ім’я виводу	Номер виводу	IO	Функції
MTMS	9	IO14	I2C_SCL
GPIO2	14	IO2	I2C_SDA

Підтримуються як ведучий, так і ведений I2C; функція інтерфейсу I2C реалізована програмно і має максимальну тактову частоту 100 кГц; тактова частота I2C повинна бути вищою, ніж мінімальна тактова частота веденого пристрою. ESP8266EX має інтерфейс вводу даних I2S та інтерфейс виводу

даних I2S; інтерфейс I2S в основному використовується для збору даних, обробки та передачі аудіоданих, послідовного вводу та виводу даних (наприклад, підтримка світлодіодів серії WS2812). Функціональність інтерфейсу I2S може бути реалізована в програмному забезпеченні за допомогою мультиплексованих GPIO і відповідного списку DMA підтримується. Опис виводів I2S наведено нижче у таблиці 2.4.

Таблиця 2.4 - Опис виводів I2S

I2S вводу даних			
Ім’я виводу	Номер виводу	IO	Функціонал
MTDI	10	IO12	I2SI_DATA
MTCK	12	IO13	I2SI_BCK
MTMS	9	IO14	I2SI_WS
MTDO	13	IO15	I2SO_BCK
UORXD	25	IO3	I2SO_DATA
GPIO2	14	IO2	I2SO_WS

ESP8266EX має два інтерфейси UART: UART0 і UART, описані нижче (табл. 2.5).

Таблиця 2.5 - Опис виводів UART

Тип виводу	Ім’я виводу	Номер виводу	IO	Функції
UART0	U0RXD	25	IO3	U0RXD
	U0TXD	26	IO1	U0TXD
	MTDO	13	IO15	U0RTS
	MTCK	12	IO13	U0CTS
UART1	GPIO2	14	IO2	U1TXD
	SD_D1	23	IO8	U1RXD

Передача даних через інтерфейс UART може бути реалізована апаратними засобами, швидкість передачі даних через інтерфейс UART може досягати 115200x40 (4,5 Мбіт/с).

UART0 можна використовувати для комунікації. Він підтримує керування потоком. UART1 має лише один сигнал передачі (Tx) і зазвичай використовується для виведення налагоджувальної інформації. ESP8266EX має 17 GPIO, які можна використовувати для підтримки різних функцій світлодіод і вікнопок. Нижче описано деякі з призначених GPIO (табл. 2.6).

Таблиця 2.6 - Спеціалізовані Функції Для розробки демонстраційних програм

Ім’я виводу	Номер виводу	IO	Функції
MTCK	12	IO13	BUTTON (RESET)
GPIO0	15	IO0	Wi-fi Light
MTDI	10	IO12	Link Light

Всього визначено три інтерфейси. Вони призначені для світлодіодів. В принципі, MTCK використовується для керування кнопкою скидання, GPIO0 використовується як сигнал для індикації стану роботи Wi-Fi і MTDI використовується як світловий сигнал для індикації стану зв'язку між пристроєм і сервером.

Електричні характеристики наведені нижче (табл 2.7).

Таблиця 2.7 - Електричні характеристики

Параметр		Стан	Мін.	Типове значення	Макс.	Од.вим.
Діапазон температур зберігання		-	-40	Кімнатна	125	ºC
Максимальна температура пайки		IPC/JEDEC J-STD-020	-	-	260	ºC
Робоча напруга		-	3.0	3.3	3.6	В
I/O	V_{IL}/V_{IH}	-	$-0.3/0.75V_{10}$	-	$0.25V_{10}/3.6$	В
	V_{OL}/V_{OH}	-	$N/0.8V_{10}$	-	$0.1V_{10}/N$	
	I_{max}	-	-	-	12	мА
Електростатичний розряд (HBM)		TAMB=25 ºC	-	-	2	кВ
Електростатичний розряд (CDM)		TAMB=25 ºC	-	-	0.5	кВ

2.5 Загальна будова системи

Підключення мікросхеми до акселерометра Далі описано підключення мікросхеми ESP8266EX до датчика MPU-6050 (рис. 2.10).

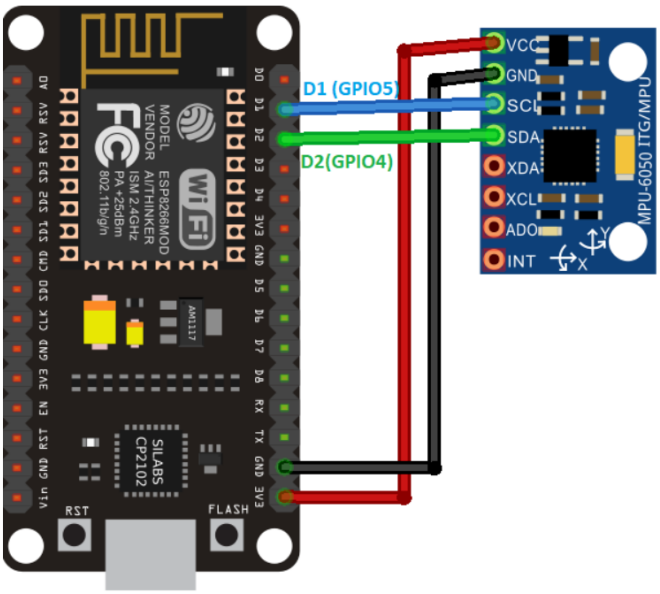


Рисунок 2.10 - Підключення ESP8266 NodeMCU з MPU-6050

На рис. 2.11 наведено структурну схему апаратно-програмного забезпечення системи збору даних.

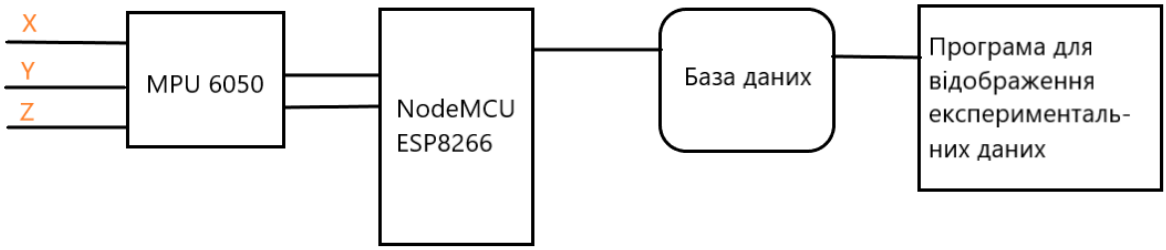


Рисунок 2.11 - Структурна схема апаратно-програмного забезпечення системи збору даних

Гіроскопи вимірюють швидкість обертання (рад/с), тобто зміну кутового положення по осях X, Y і Z (крен, тангаж і рискання) з часом. Це дозволяє визначити орієнтацію об'єкта. Акселерометри вимірюють прискорення (швидкість зміни швидкості об'єкта). Вони виявляють статичні сили, такі як сила тяжіння (9,8 м/с2) і динамічні сили, такі як вібрація і рух. MPU-6050 вимірює прискорення по осях X, Y і Z. В ідеалі, статичний об'єкт повинен мати прискорення, що дорівнює силі тяжіння по осі Z і нульове прискорення по осях X і Y.

Використовуючи значення акселерометра, кути крену і тангажу можна обчислити за допомогою тригонометрії. Однак, кут рискання обчислити неможливо. Розглянемо положення осей, які можуть бути виміряні датчиками (рис. 2.12).

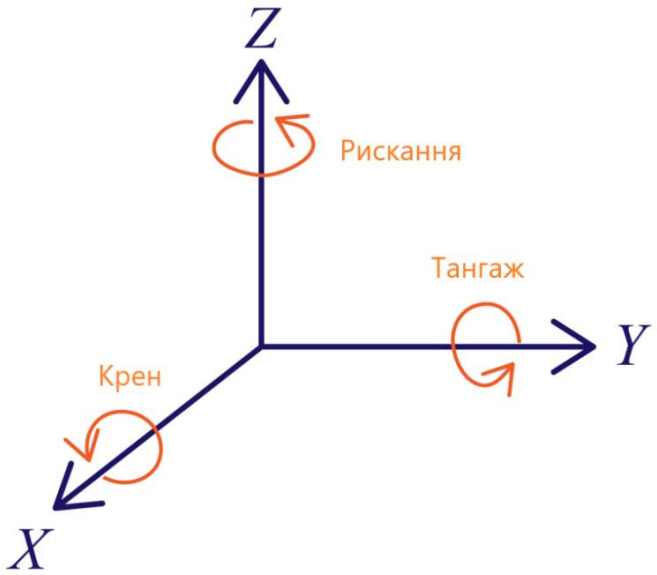


Рисунок 2.12 - Зображення осей за якими відбувається вимірювання показників акселерометра MPU-6050

Мікросхема NodeMCU ESP8266 виконує роль обробника подій, після чого коректна інформація надсилається в базу даних для зберігання, а спеціальна програма відображає цю інформацію у визначеному користувачем форматі.

2.6. Інтерфейс I2C

Опишемо які є призначення та принцип роботи у даного інтерфейсу. Сам стандарт досить старий, він був винайдений інженерами Philips більше 30 років тому. З цієї причини він дуже широко використовується в електронних пристроях різного призначення, різними модифікаціями і під різними назвами ; в середовищі Arduino це, мабуть, другий за поширеністю інтерфейс після UART і найчастіше використовується для обміну даними між контролерами, датчиками і виконавчими пристроями. Позитивним моментом є те, що на відміну від UART, він має вищу і стабільнішу швидкість передачі, що

забезпечує більш стабільну і швидку передачу даних. Його архітектура також дозволяє підключати до 127 пристроїв одночасно до однієї шини, що складається з двох ліній, SDA (дані) і SCL (тактова частота), без додаткового обладнання, крім двох підтягуючих резисторів. Нижче описана будова шини I2C (рис. 2.13).

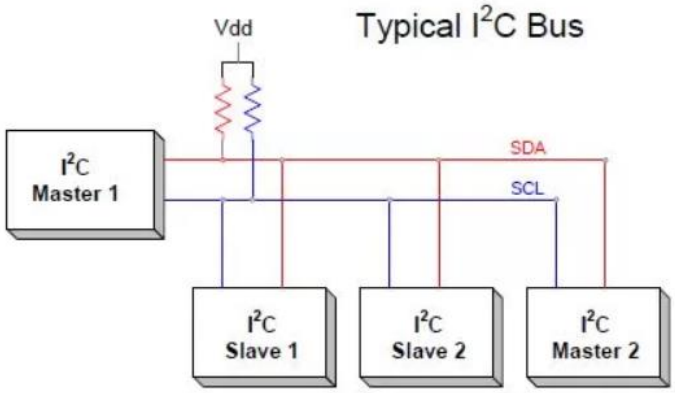


Рисунок 2.13 - Будова шини I2C

На відміну від UART, i2c є синхронним протоколом зв'язку і використовує для обміну даними синхронні сигнали, спільні для всіх підключених пристроїв. Лише один головний пристрій, який називається Master, відповідає за генерування сигналів. Цей пристрій "розмовляє" або "передає слова" іншим пристроям, які англійською мовою називаються Slaves. Кожен керований пристрій має унікальний номер, навіть якщо у всій мережі є лише один такий пристрій. Майстер не має номера, і всі його знають. Мережа завжди в ідеальному порядку, тому що підлеглі мовчки слухають, що говорить ведучий, і відповідають лише тоді, коли ведучий називає їх на ім'я (номер) і вимагає цього.

У типовому застосуванні інтерфейсу I2C підключені пристрої можуть бути ведучими або веденими. Ведучий встановлює адресу веденого пристрою на шині, ведений пристрій з відповідною адресою відповідає адресі ведучого, а ведений пристрій з відповідною адресою підтверджує її. Коли MPU-60X0 зв'язується з системним процесором, який є ведучим пристроєм, що діє як ведений пристрій; лінії SDA і SCL зазвичай вимагають підтягуючого резистора до VDD. Максимальна швидкість шини - 400 кГц Адреса веденого пристрою

MPU-60X0 - b110100X довжиною 7 біт. Біт LSB 7-бітної адреси визначається логічним рівнем виводу AD0. Це дозволяє підключити два MPU-60X0 до однієї шини I2C. При використанні в такій конфігурації адреса одного пристрою - b1101000 (логічний низький рівень на виводі AD0), а адреса іншого пристрою - b1101001 (логічний високий рівень на виводі AD0).

Головний пристрій допоможе правильно використовувати його, маючи загальне розуміння того, як працює протокол I2C, що відбувається всередині пристрою і в якій послідовності. Майстер починає роботу зі зниження рівня шини SDA, що є сигналом запуску веденого пристрою. Підвищення рівня SDA високим тактовим сигналом є командою зупинки для підлеглого. Все, що відбувається під час цих подій, називається повідомленням, тобто передачею даних (рис 2.14).

Рисунок 2.14 - Передача даних по шині I2C

Протокол зв'язку I2C має стани START (S) і STOP (P). Зв'язок на шині I2C ініціюється, коли ведучий встановлює стан СТАРТ (S) шини. Цей стан визначається як перехід лінії SCL у високий рівень, тоді як лінія SDA переходить з високого рівня в низький (див. Діаграму нижче). Шина вважається зайнятою до тих пір, поки ведучий не встановить стан STOP (P) на шині; стан STOP (P) – це перехід лінії SDA з низького рівня на високий, в той час як лінія SCL знаходиться на високому рівні.

Також, якщо замість умови STOP повторно генерується умова START (Sr), шина залишається зайнятою Байт даних I2C має довжину 8 біт; немає обмежень на кількість байтів, які можуть бути передані за одну передачу даних. Кожен переданий байт повинен супроводжуватися сигналом підтвердження (ACK). Тактовий сигнал підтвердження генерується ведучим пристроєм, а

приймач генерує свій власний сигнал підтвердження, опускаючи SDA і утримуючи його на низькому рівні, коли тактовий сигнал підтвердження знаходиться на високому рівні.

Якщо ведений пристрій зайнятий і не може відправити або отримати наступний байт даних, поки не завершиться інша задача, ведений пристрій може утримувати низький рівень SCL і таким чином змусити ведучий пристрій чекати. Коли ведений пристрій готовий і лінія синхронізації звільняється, нормальна передача даних відновлюється (див. схему нижче).

Після ініціювання обміну даними за умови START (S), ведучий передає 7-бітну адресу веденого, за якою слідує восьмий біт читання/запису. Біт читання/запису вказує, чи приймає ведучий дані від веденого, чи записує дані до веденого. Після цього ведучий звільняє лінію SDA і чекає на сигнал підтвердження (ACK) від веденого.

(ACK) для підтвердження передачі. Кожен переданий байт повинен супроводжуватися бітом підтвердження. Для підтвердження ведений пристрій тягне лінію SDA на низький рівень і утримує його, поки SCL знаходиться на високому рівні. Передача даних завжди припиняється ведучим за умов STOP (P) і лінія зв'язку звільняється. Однак, ведучий може створити другу умову START (Sr) і звернутися до іншого веденого без попереднього створення умови STOP (P); умова зупинки визначається, коли лінія SDA переходить з низького рівня в високий, коли SCL знаходиться в стані High. За винятком умов запуску і зупинки, всі зміни SDA повинні відбуватися при низькому рівні SCL.

При записі внутрішніх регістрів MPU-60X0, ведучий надсилає умову запуску (S), за якою слідує адреса I2C та біт запису (0). На такті 9 (коли рівень синхронізації високий), MPU-60X0 підтверджує передачу. Потім ведучий встановлює адресу регістра (RA) на шині; після того, як MPU-60X0 підтверджує отримання адреси регістра, ведучий виводить дані регістра на шину. Після цього видається сигнал ACK, і передача даних може бути припинена відповідно до умови зупинки (P). Щоб записати кілька байт після останнього сигналу ACK, ведучий може продовжувати виводити дані замість

того, щоб посилати сигнал зупинки. У цьому випадку MPU-60X0 автоматично збільшує адресу регістра і завантажує дані у відповідний регістр (рис.2.15).

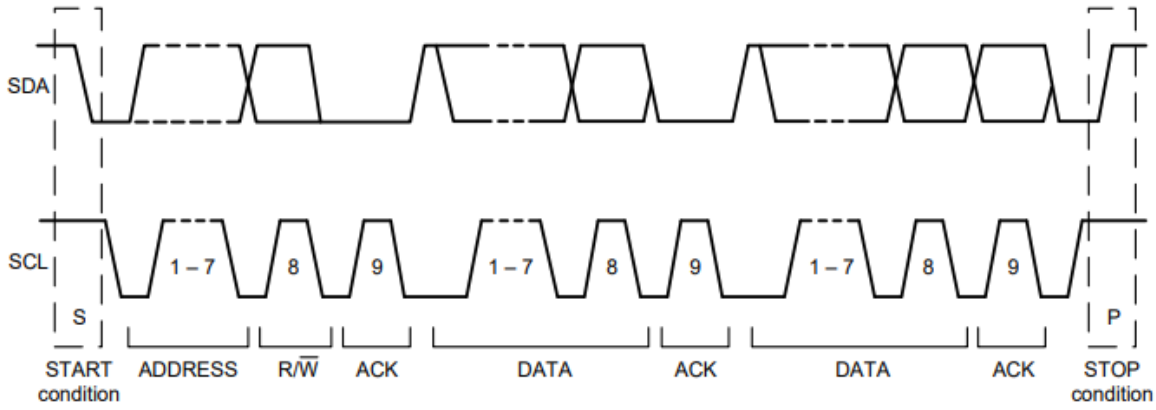


Рисунок 2.15 – Передача даних по інтерфейсу I2C

При читанні внутрішніх регістрів MPU-60X0, ведучий передає умову запуску, за якою слідує адреса I2C, біт запису та адреса регістра, який потрібно прочитати. Після отримання сигналу ACK від MPU-60X0, ведучий передає стартовий сигнал, за яким слідує адреса веденого пристрою та біт читання.

MPU-60X0 передає сигнал ACK і дані. Зв'язок завершується сигналом неприйняття (NACK) і стоп-бітом від ведучого; умова NACK визначається таким чином, щоб лінія SDA залишалася в стані високого рівня.

На рис.2.16 наведена орієнтація осей чутливості та полярність обертання для MPU-60X0.

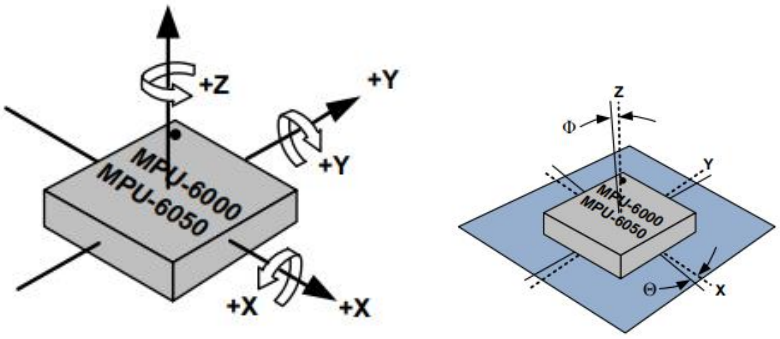


Рисунок 2.16 – Орієнтація осей чутливості та полярність обертання

На перший погляд, структура повідомлення (рис 2.17) здається набагато складнішою, ніж у звичайного UART.

Рисунок 2.17 - Блок-схема повідомлення, що надсилається за протоколом I2C

Адреса 7-10 біт (залежно від типу протоколу). Це ідентифікатор конкретного підлеглого пристрою, до якого звертається ведучий. Потім біти читання або запису описують, чи будуть дані передаватися або прийматися ACK/NACK - біти, які підтверджують або відхиляють прийом інформації в кожному кадрі. Іншими словами, ACK - це єдиний біт, який генерується веденим пристроєм на лінії SCL. На жаль, без додаткових запобіжних заходів така передача інформації є непридатною для використання і роз'єднується лише тоді, коли приймаючий пристрій перестає відповідати, скорочуючи час передачі.

Інтерфейс I2C призначений для досить швидкої та надійної передачі на невеликі відстані, зазвичай у межах одного пристрою. Як правильно, довжина дротів обмежена кількома метрами. Він займає свого роду проміжне положення між UART і SPI за параметрами швидкості, надійності, відстані та використання ресурсів.

Як говорилося вище, найчастіше протокол I2C використовується для спілкування контролера (як майстра) з датчиками та виконавчими пристроями. Це зручно, коли потрібна висока швидкість та надійність обміну даними. Наприклад, індикатори, дисплеї, годинники реального часу, датчики температури, вологості та інших параметрів повітря та інших середовищ, GPS-приймачі, RFID-рідери та багато інших воліють цей спосіб спілкування іншим. Найчастіше під кожен пристрій написана власна бібліотека, яка непомітно включає описані вище функції протоколу, але буває і так, що доводиться

розбиратися з пристроєм самостійно. Тоді беремо в руки даташит і майструємо на його основі команди для передачі та прийому даних.

Не завжди адреса пристрою відома заздалегідь, її просто забувають вказати виробники, особливо часто цим грішать китайські виробники. У такому випадку використовуємо готову, наявну в прикладі до бібліотеки Wire програму i2c_scanner, яка простим перебором обчислює адреси підключених пристроїв, що відгукуються.

Однак, протокол цілком успішно можна застосовувати для обміну даними між контролерами. Зрозуміло, один із них, згідно з доктриною, буде головним, а інший чи інші, веденими.

2.7 Розробка макету та програмного забезпечення для збору даних

Для перевірки з’єднання середовища розробки Arduino та мікроконтролера спробуємо блимнути діодом за допомогою наступного коду (рис 2.18).

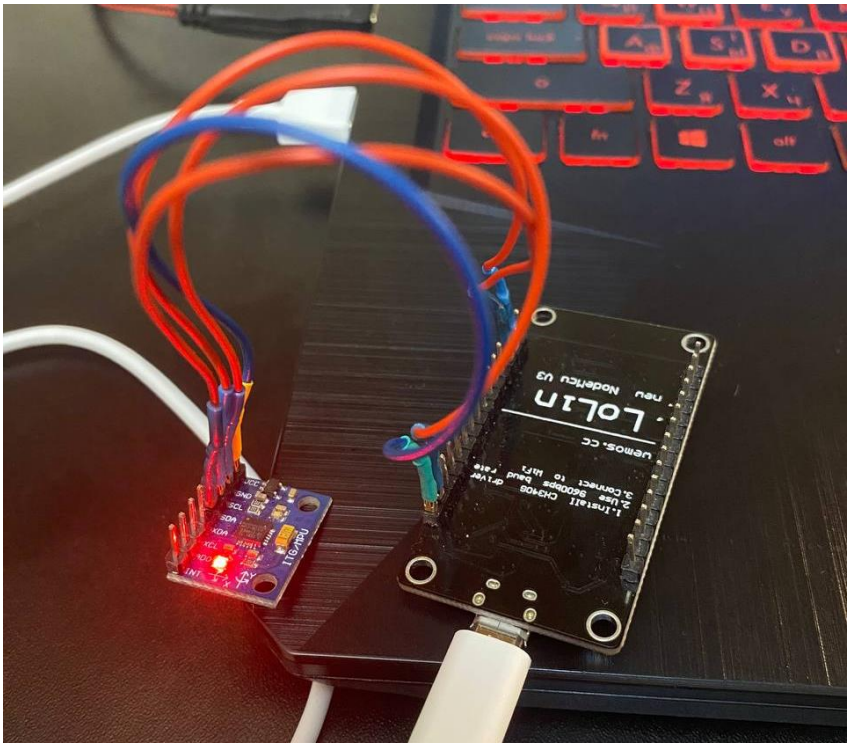


Рисунок 2.18 – Загальний вигляд макету

Функція setup(): встановлюється початковий стан для підготовки до роботи. У цьому випадку, встановлюється режим виходу (OUTPUT) для піна, який буде використовуватися для підключення світлодіода до мікроконтролера.

```
void setup() {
      pinMode(pin, OUTPUT);
}
```

Функція loop(): ця функція викликається постійно після завершення setup(). Вона містить команди для блимання світлодіода:

- digitalWrite(pin, HIGH): Встановлює високий рівень напруги на піні pin, увімкнувши світлодіод.
- delay(1000): Затримка на 1000 мілісекунд (1 секунда).
- digitalWrite(pin, LOW): Встановлює низький рівень напруги на піні pin, вимикаючи світлодіод.
- delay(1000): Знову затримка на 1000 мілісекунд (1 секунда).

```
void loop() {
      digitalWrite(pin, HIGH);
      delay(1000);
      digitalWrite(pin, LOW);
      delay(1000);
}
```

У результаті отримаємо успішне виконання коду (рис. 2.19) та блимання діода на платформі ESP8266 (рис. 2.20).

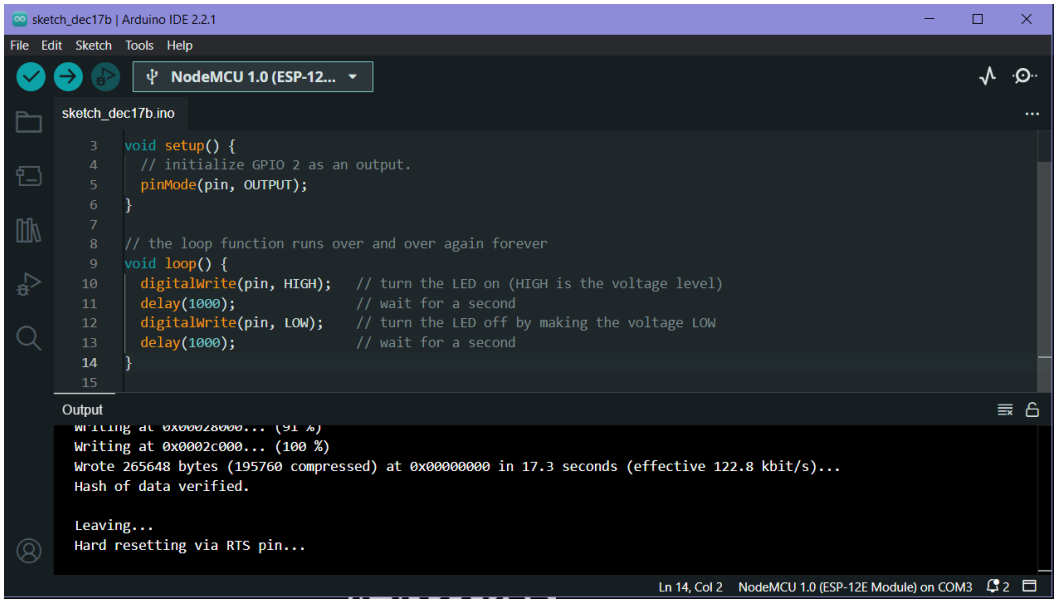


Рисунок 2.19 - Завантаження програмного забезпечення

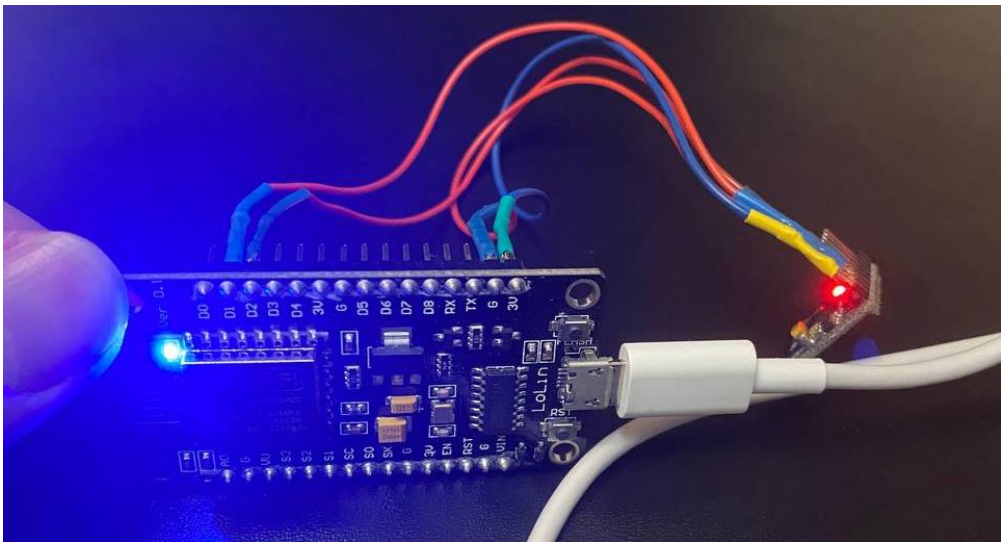


Рисунок 2.20 - Перевірка працездатності макету

2.8 Калібрування датчика MPU6050

Калібрування датчика, такого як MPU6050, важливо з двох основних причин:

1. Підвищення точності вимірювань: датчики можуть мати фабричні відхилення, систематичні помилки або неоднаковості в характеристиках, що можуть призводити до неточних вимірювань.

Калібрування дозволяє скоригувати ці неточності, щоб отримати більш точні результати вимірювань.

2. Забезпечення нормальної роботи системи: у багатьох системах, де використовуються датчики, точність вимірювань є критичною. Наприклад:

- Навігаційні системи: для точного визначення положення та орієнтації пристроїв, таких як дрони, роботи з віртуальною реальністю (VR) або допоміжне обладнання у навігації автономних автомобілів.
- Медичне обладнання: датчики використовуються для вимірювання рухів пацієнтів чи стану їх тіла, де висока точність є важливою для правильної діагностики або лікування.
- Промислове обладнання: у виробництві і автоматизованих системах точність вимірювань може впливати на якість виробу або безпеку процесів.

Наприклад, навігаційна система для дрона, яка використовує датчик MPU6050 для вимірювання обертання та прискорення, може видавати неточні результати через фабричні помилки. Це може призвести до неправильного розташування або навіть аварійного зіткнення дрона. Калібрування датчика допоможе скоригувати ці помилки і забезпечити більш точну навігацію.

Навіть невеликі помилки у датчику можуть мати величезний вплив на результати вимірювань у важливих системах, тому калібрування є важливою процедурою для досягнення високої точності та надійності даних, що отримуються від датчиків.

Отже, перейдемо до прецесу калібрування. Для цього використаємо бібліотеку MPU6050_tockn та наступний код.

Підключаються бібліотеки MPU6050_tockn та Wire для роботи з датчиком і забезпечення зв'язку по I2C. Створюється об'єкт mpu6050 для взаємодії з датчиком.

```
void loop(){}

#include <MPU6050_tockn.h>
```

```
#include <Wire.h>

MPU6050 mpu6050(Wire);
```

У функції setup() ініціалізується з'єднання по I2C (Wire.begin()) та запускається датчик (mpu6050.begin()). Функція mpu6050.calcGyroOffsets() викликається для автоматичного обчислення і калібрування зміщень (offsets) гіроскопа. Це дозволяє скоригувати систематичні помилки гіроскопа для отримання більш точних вимірювань обертання.

```
void setup(){
  Wire.begin();
  mpu6050.begin();
  mpu6050.calcGyroOffsets();
}
```

У функції loop() зазвичай знаходиться основний код програми, який виконується постійно після завершення setup(). У даному прикладі функція loop() пуста, оскільки основна робота (калібрування) відбувається під час ініціалізації у функції setup().

Результатом виконання даної програми (рис. 2.21) буде автоматично відкалібрований датчик, що дозволить отримувати максимально точні значення гіроскопа, адже від точності даних, які надходять та чіткості роботи датчика буде залежати вся подальша робота системи збору даних.

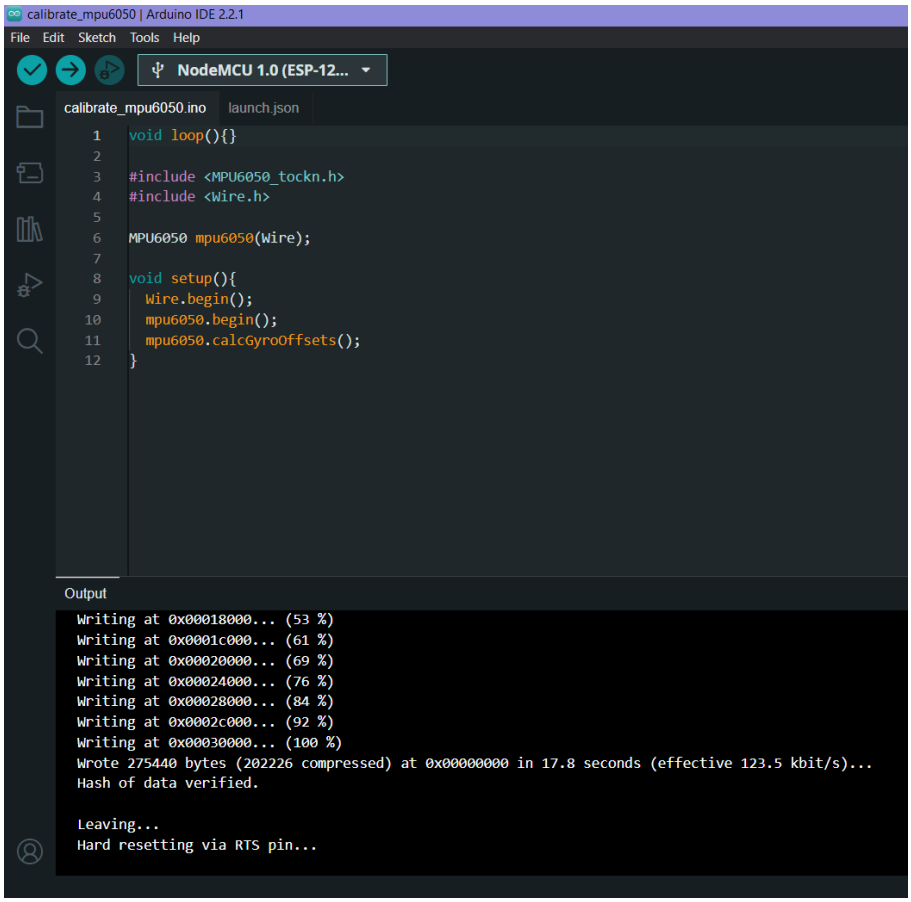


Рисунок 2.21 - Виконання калібрування датчика

Бібліотека `MPU6050_tockn дозволяє спростити процес роботи з датчиком MPU6050 в середовищі Arduino. Вона забезпечує готові функції для калібрування датчика. Основні кроки для калібрування датчика MPU6050 за допомогою цієї бібліотеки в середовищі Arduino можуть виглядати так:

1. Підключення датчика до Arduino: з'єднання датчика MPU6050 з Arduino за допомогою зв'язку I2C. Підключення проводиться за допомогою відповідних пінів (SCL, SDA) на Arduino до відповідних пінів на датчику.

2. Встановлення бібліотеки `MPU6050_tockn`: у середовищі Arduino вибераємо "Sketch" -> "Include Library" -> "Manage Libraries", знайдемо бібліотеку `MPU6050_tockn` та встановлюємо її.

3. Використання функцій бібліотеки для калібрування: бібліотека надає функції для автоматичного калібрування датчика. Це може включати функції для зчитування даних з датчика, обробки цих даних для визначення зсувів чи систематичних помилок, а потім застосування корекційних параметрів.

4. Отримання корекційних параметрів: після виконання калібрування бібліотека може повернути корекційні параметри, такі як зміщення або масштабування даних, які можна застосувати для подальших вимірювань.

5. Використання скорегованих даних: отримані корекційні параметри можна використовувати для отримання точних даних з датчика MPU6050 у майбутніх програмах або проектах.

Програмне забезпечення для калібрування датчика MPU-6050 наведено у додатку А.2.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

3.1 Розробка програмного забезпечення рівня збору даних

Розроблено програму для отримання данних з датчика MPU-6050: акселерометра, гіроскопа та температури. Він отримує кутову швидкість (гіроскоп) на осях x, y і z, прискорення на осях x, y і z і температуру. Для тестового підключення використано середовище Arduino.

Почнемо з включення необхідних бібліотек для датчика MPU-6050: Adafruit_MPU6050 і Adafruit_Sensor.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
```

Створюємо файл Adafruit_MPU6050 об'єкт під назвою МПУ для поводження з датчиком.

```
Adafruit_MPU6050 mpu;
```

У файлі setup(), ініціалізуємо послідовний монітор зі швидкістю передачі даних 115200.

```
Serial.begin(115200);
```

Ініціалізуємо датчик MPU-6050.

```
if (!mpu.begin()) {
  Serial.println("Sensor init failed");
  while (1)
    yield();
}
```

Встановлюємо діапазон вимірювання акселерометра:

```
mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
```

Встановлємо діапазон вимірювання гіроскопа:

```
mpu.setGyroRange(MPU6050_RANGE_500_DEG);
```

Встановлюємо пропускну здатність фільтра:

```
mpu.setFilterBandwidth(MPU6050_BAND_5_HZ);
```

У файлі loop() ми отримаємо показання датчиків і відобразимо їх на послідовному моніторі.

По-перше, потрібно отримати нові події датчика з поточними показаннями.

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
```

Нарешті, роздрукуємо показання. Для прискорення:

- a.acceleration.x: отримує прискорення по осі x;
- a.acceleration.y: отримує прискорення по осі y;
- a.acceleration.z: отримує прискорення по осі z.

Прискорення вимірюється в метрах на секунду квадратних (м/с2)

```
Serial.print("Acceleration X: ");
Serial.print(a.acceleration.x);
Serial.print(", Y: ");
Serial.print(a.acceleration.y);
Serial.print(", Z: ");
Serial.print(a.acceleration.z);
Serial.println(" m/s^2");
```

Щоб отримати показання гіроскопа:

- g.gyro.x: отримує кутову швидкість на осі x;
- g.gyro.y: отримує кутову швидкість на осі y;
- g.gyro.z: отримує кутову швидкість на осі z.

Кутова швидкість вимірюється в радіанах за секунду (рад/с).

```
Serial.print("Rotation X: ");
Serial.print(g.gyro.x);
Serial.print(", Y: ");
Serial.print(g.gyro.y);
Serial.print(", Z: ");
Serial.print(g.gyro.z);
Serial.println(" rad/s");
```

Наостанок виводимо температуру – вона вимірюється в градусах Цельсія.

```
Serial.print("Temperature: ");
Serial.print(temp.temperature);
Serial.println(" degC");
```

Нові показання датчика відображаються кожні 10 мілісекунд.

```
delay(10);
```

У результаті роботи коду та переміщення датчика отримаємо такі результати (рис 3.1).

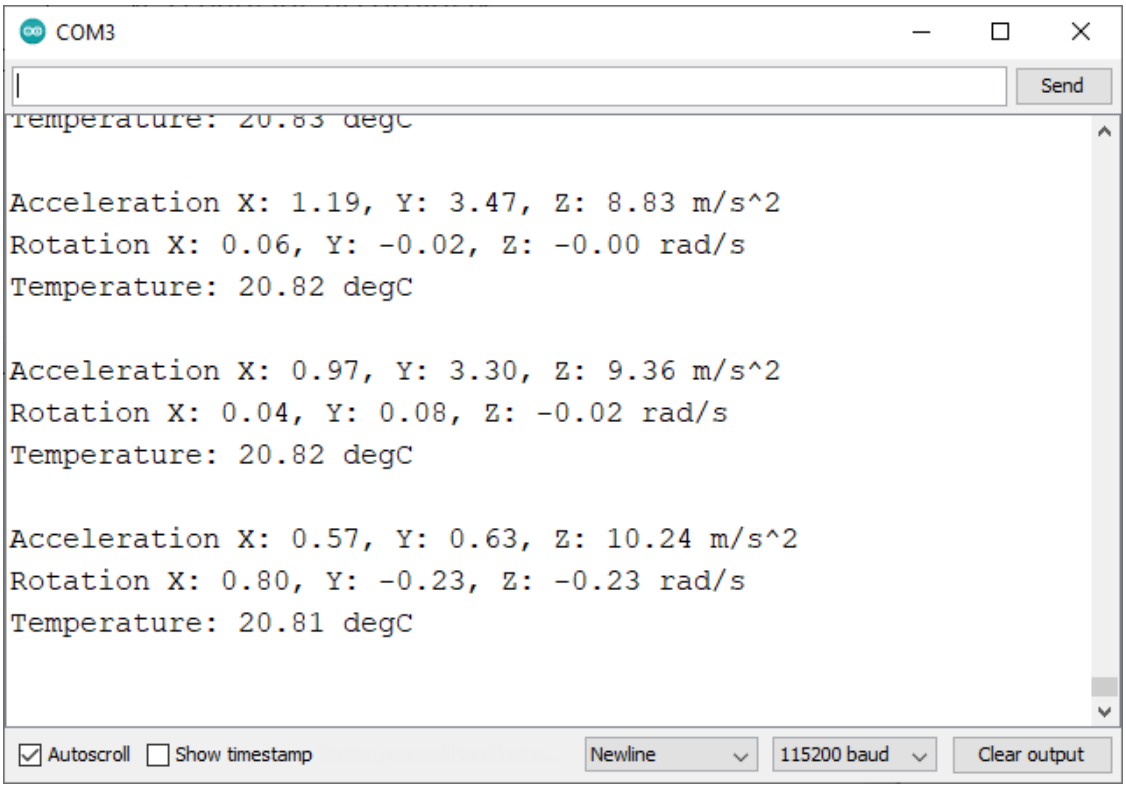


Рисунок 3.1 - Отримання даних з датчика MPU-6050

3.2 Розробка серверного програмного забезпечення на мові Python

Розроблено програму на мові Python для обробки даних, що отримуються з датчика MPU-6050 з використанням бібліотеки Quis Streams. Для тестового підключення використано середовище Pycharm.

Основною структурою, яка використовується для організації даних у Quix, є тема. Теми часто розбиваються на кілька потоків, щоб забезпечити горизонтальне масштабування. Оскільки до теми може писати декілька пристроїв або джерел, кожне джерело пише у власний потік, щоб забезпечити масштабування та організацію повідомлень. Пристрій 1 пише до потоку 1, пристрій 2 - до потоку 2 і так далі. Це ідея потокової контекстної IoT-телеметрії.

Quix Streams забезпечує збереження контексту потоку. Це означає, що повідомлення в межах одного потоку завжди публікуються в одному розділі. Це означає, що повідомлення в межах одного потоку завжди публікуються в одному і тому ж розділі. Це означає, що в межах одного потоку споживачі

можуть довіряти порядку повідомлень. Розділ може містити декілька потоків, але потік завжди обмежений одним розділом.

Код, який обробляє потоки в межах теми, організований з використанням ідеї груп споживачів. Це вказує брокеру, що він буде обробляти тему з усіма доступними репліками в групі споживачів і розділятиме обробку всіх потоків в темі. Коли кілька реплік розміщено в групі споживачів, горизонтальне масштабування відбувається автоматично, оскільки потоки розподіляються між репліками. Наприклад, якщо є три потоки і три репліки, кожна репліка обробляє один потік.

Імпортуємо всі необхідні бібліотеки та їх компоненти:

```
import time
import datetime
from quixstreams import *
```

Для початку публікації даних з використанням Quix спочатку створено екземпляр:

```
client = KafkaStreamingClient('127.0.0.1:9092')
```

Для публікації даних у темі створеноо екземпляр:

```
topic_producer = client.get_topic_producer(TOPIC)
```

Створено потік для передачі даних:

```
stream = topic_producer.create_stream()
```

Реєструємо зворотній викликі потоку, який викликається при першому отриманні даних у потоці. Коли потік стає активним, він реєструє зворотний виклик для обробки даних часових рядів у форматі кадру даних.

```
      def on_stream_received_handler(stream_consumer: qx.StreamConsumer):
          stream_consumer.timeseries.on_dataframe_received =
on_dataframe_received_handler
      topic_consumer.on_stream_received = on_stream_received_handler
```

Фіксуємо кожні 100 оброблених повідомлень або раз у 500 мс, залежно від того, що настане раніше.

```
    commit_settings = CommitOptions()
    commit_settings.commit_every = 100
    commit_settings.commit_interval = 500
    commit_settings.auto_commit_enabled = True
    topic_consumer = client.get_topic_consumer(`stream_data`,
commit_settings=commit_settings)
```

Далі реалізовано публікацію даних у потоки:

```
with (topic_producer := client.get_topic_producer(TOPIC_ID)):
    stream = topic_producer.create_stream()
```

Бібліотека надає можливість додати необов'язковий контекст до своїх потоків, додавши назву, деякі метадані або розташування за умовчанням. Ця назва є відображуваною назвою потоку на платформі. Quix Platform використовуватиме його замість ідентифікатора потоку, щоб представити ваш потік на платформі. Вкажемо цю назву:

```
stream.properties.name = "Stream data from MPU-6050"
```

Проведемо підготовку даних та опублікуємо дані, після цього закривши поток:

```
for index in range(0, 3000):
    stream.timeseries \
        .buffer \
        .add_timestamp(datetime.datetime.utcnow()) \
        .add_value("Acceleration", index) \
        .add_value("Rotation", index) \
        .add_value("Temperature", index) \
        .publish()
    time.sleep(0.01)
print("Closing stream")
stream.close()
```

Quix Streams надає можливість публікувати та підписуватися на необроблені, неформатовані повідомлення, а також працювати з ними як байтами. Це дає нам можливість реалізовувати протокол за потреби та конвертувати між форматами. Опублікуємо необроблені повідомлення:

```
with (producer := client.get_raw_topic_producer(TOPIC_ID)):
    data = bytearray(bytes("TEXT CONVERTED TO BYTES",'utf-8'))

    message = RawMessage(data)
    message.key = MESSAGE_KEY
    producer.publish(message)
```

Одна з проблем, яка може виникнути, полягає в тому, що дані в пам'яті не зберігаються під час перезапусків, вимкнень і збоїв екземпляра. Це можна пом'якшити, використовуючи об'єкт Quix Streams. Це гарантує, що вказані

змінні зберігатимуться в постійному сховищі, і ці дані зберігатимуться під час перезапусків, вимкнень і збоїв системи. Це гарантує, що змінна зберігається, оскільки періодично (за замовчуванням 20 секунд) вона скидається до локального файлового сховища.

Якщо система виходить з ладу (або перезавантажується), Kafka відновлює обробку повідомлень з останнього збереженого повідомлення. Цей об'єкт вбудований в Кафку.

Додано використання LocalFileStorage для збереження даних під час перезапуску системи та збоїв:

```
g_running_total_per_stream = qx.InMemoryStorage(qx.LocalFileStorage())

def callback_handler (stream_consumer: qx.StreamConsumer, data:
qx.TimeseriesData):
    if stream_consumer.stream_id not in g_running_total_per_stream:
        g_running_total_per_stream[stream_consumer.stream_id] = 0

g_running_total_per_stream[stream_consumer.stream_id] += some_value

def on_stream_received_handler(stream_consumer: qx.StreamConsumer):
    stream_consumer.timeseries.on_data_received = callback_handler

topic_consumer.on_stream_received = on_stream_received_handler
topic_consumer.on_committed = g_running_total_per_stream.flush
```

Отримавши інстанс, починаємо отримувати потоки. Для кожного потоку, отриманого за вказаною темою, буде виконуватися зворотний виклик,який викликається щоразу, коли отримуватиме новий потік. Функція subscribe() починає споживати потоки та дані з визначеної теми. Важливо робити це лише після того, як зареєстровані зворотні виклики для всіх подій, які необхідно прослухати.

```
topic_consumer.on_stream_received = on_stream_received_handler
topic_consumer.subscribe()
```

За допомогою параметру StreamEndType визначаємо причину закриття потоку, це додано у функцію зворотного виклику:

```
def on_stream_closed_handler(stream: StreamConsumer, end_type:
StreamEndType):
        print("Stream closed with {}".format(end_type))
```

На терміналі показано виведення даних, що записуються через нашу програму (рис. 3.2):

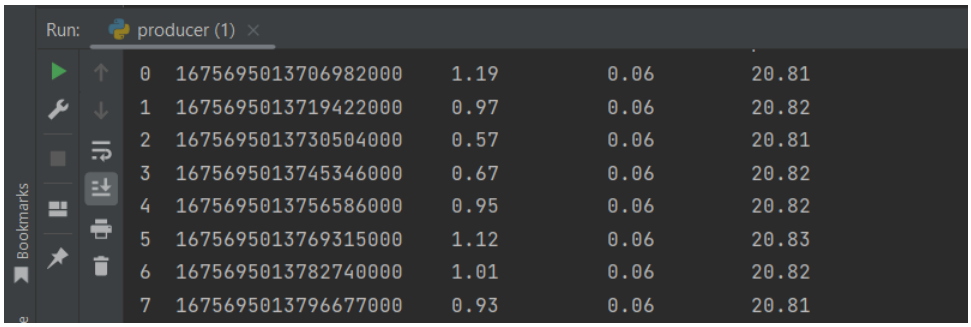


Рисунок 3.2 - Приклад отримання данних з датчика

На рис 3.3 наведено графічний вивід отриманих значень за допомогою платформи Quix Streams.

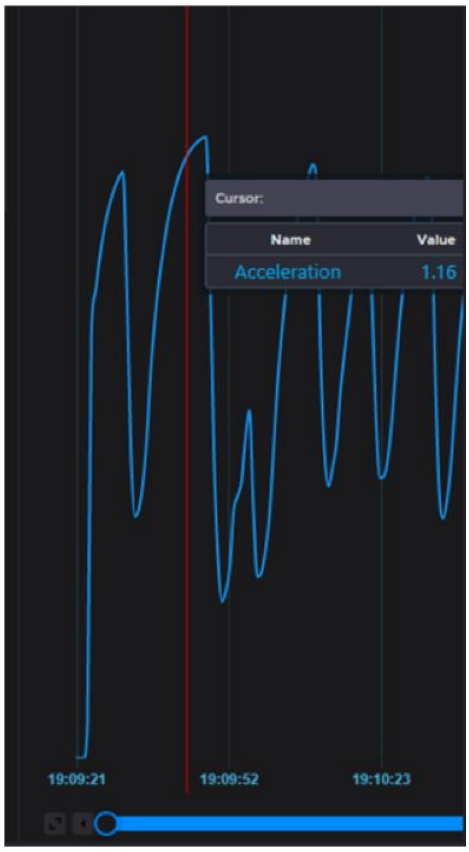


Рисунок 3.3 - Графік отриманих даних на платформі Quix Streams

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ «СИСТЕМА ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ НА ОСНОВІ ПОТОКОВИХ ТЕХНОЛОГІЙ»

4.1 Опис ідеї та технологічний аудит стартап-проекту

Інтернет речей дуже швидко розвивається останнім часом, та створюється все більше нових технологій, що допомагають людству у різних сферах життя – таких як обслуговування різного роду приладів, досить багато уваги приділяється медицині, спорту, не мало нових технологій було впроваджено у системах обслуговування клієнтів, та багато інших сфер діяльності людини, де інтернет речей відіграє не останню роль.

Всі вони мають одну спільну рису, правильна взаємодія з якою відіграє ключову роль у використанні цих технологій – наявність певної кількості даних. Саме дані розповідають нам про те, що клієнти споживають, по яким цінам вони купують, які товари краще продаються, які ліки слід порекомендувати лікарю при певних хворобах, і які це хвороби по симптомам. Саме правильно зберігати, обробляти та у подальшому використовувати надану інформацію є визначною рисою кожної системи.

Проблемою даних систем можна назвати те, що більшість із них адаптовані під використання певних мов програмування, таких як Java, Scala, оскільки вони вважаються більш придатними для потокової обробки через свої особливості та швидкодію. Однак, використання Python для потокової обробки даних у реальному часі залишається викликом через його не оптимальність та обмеження у цій сфері.

Саме даний стартап-проект вирішує проблему використання мови Python для потокової обробки даних, адже він використовує механізми, що дозволяють уникнути основних проблем, пов'язаних із розробкою такого програмного забезпечення.

Основними характеристиками такої системи, можна назвати таке – висока пропускна здатність, що визначає кількість даних, яку система може обробляти

протягом певного часу, адже вона дозволяє ефективно обробляти великі потоки даних без затримок; масштабованість, що є здатністю системи збільшувати обсяг обробки даних з ростом їхнього обсягу, бо система повинна бути здатна адаптуватися до збільшення навантаження без втрати продуктивності; надійність, адже система має захист від втрати даних, використовувати механізми відновлення та управління помилками; якісна оптимізація ресурсів, де використання пам'яті та обчислювальних потужностей є важливим для ефективної обробки великих потоків даних; паралельність та розподіленість, що надає можливість обробки багатьох операцій паралельно та розподілено у мережі, та зможе дозволити прискорити обробку даних та забезпечити високу швидкість великих потоків; моніторинг та аналіз, який забезпечує систему інструментами для моніторингу та аналізу великих потоків даних, що дозволяє вчасно виявляти проблеми та оптимізувати роботу системи.

Висновки: стартап-проект, що розроблюється, має всі необхідні інструменти для якісної обробки даних, які відповідають усім сучасним вимогам у сфері інтернету речей та дозволяє нівелювати всі труднощі у розробці програмного забезпечення на мові Python у порівнянні з його конкурентами. Простота та популярність мови Python дозволяє ще більш широко використовувати її у всіх сферах життя. Тому розробка такого стартап-проекту має досить високу актуальність у будь-якій сфері, яка вимагає обробки великого обсягу даних, що одночасно потрапляють до системи.

Розглянемо переваги та недоліки даного стартап-проекту (табл. 4.1).

Таблиця 4.1 Визначення переваг та недоліків

Параметр	Python з Quix Streams	Аналоги
Ефективність	Працює просто та ефективно	Apache Kafka, який, хоч і швидкий, може вимагати додаткових налаштувань для роботи з Python

Масштабованість	Має просту та зрозумілу масштабованість	Spark Streaming може виявляти обмеження у масштабуванні для дуже великих обсягів даних
Підтримка іншого обладнання	Підтримує всі існуючі популярні пристрої	Storm може мати обмежену підтримку для деяких пристроїв
Надійність	Надійна та відмовостійка система	Apache Samza може мати проблеми зі стабільністю при великому навантаженні
Відкритий код	Має відкритий код та доступний до перегляду	Microsoft Azure Stream Analytics, який може мати обмежені можливості налаштування через пропріетарний характер
Простота використання	Вимагає знання мови Python та базового розуміння про потоки даних	Apache Flink (Java) потребує розуміння Java та Scala

Висновки: Розроблена система, заснована на Python з використанням Quix Streams, виявляється привабливою альтернативою у світі потокової обробки даних. Вона вирізняється простотою використання та здатністю масштабуватися з ростом обсягу даних, маючи перевагу у підтримці різноманітних пристроїв. У порівнянні з аналогами, забезпечує зручність в розробці та надійну обробку даних на Python, хоча деякі аналоги можуть виявити обмеження у масштабуванні та потребувати додаткових налаштувань для використання з Python.

Технологічний аудит - це метод оцінки технічного стану компанії, перевірки організації всіх ключових процесів за визначеними критеріями, виявлення переваг та недоліків виробництва.

У таблиці 4.2 проведемо аудит системи, за допомогою якої буде реалізовуватись стартап.

Таблиця 4.2 Технологічна здійсненність ідеї проекту

1	2	3	4	5
№ п/п	Основна ідея	Спосіб реалізації	Наявність технологій, що будуть використовуватись	Доступність технологій, що будуть використовуватись
1	Високоефективна система збору потокових даних	Аналіз доступних систем та способів обробки даних	Наявна	Доступна
2	Збір експериментальних даних у потоковому режимі	Розробка системи, що забезпечує потрібний рівень технологічності	Наявна	Доступна
3	Обробка даних	Інтеграція програмного забезпечення у системи інтернет речей	Наявна	Доступна

Технологія реалізації та впровадження ідеї стартапу- створення системи збору потокових даних, що буде ефективно опрацьовувати та зберігати велику кількість даних, забезпечуючи загальну відмовостійкість та пропускну здатність

Висновки: Проаналізувавши таблицю, робимо такий висновок, що даний стартап-проект можемо використовувати як у реальних наукових проектах так і у бізнес задачах, у всіх сферах, де використовуються та оброблюються будь-які дані.

4.2 Аналіз ринкових можливостей запуску стартап проекту

Щоб проаналізувати ринкові можливості продукту, необхідно оцінити його за наступними критеріями: кількість конкурентів, загальний можливий обсяг продажу, динаміка ринку, рентабельність та присутність специфічних умов ліцензування продукту. Опишемо ці критерії у таблиці 4.3.

Таблиця 4.3 Попередня характеристика потенційного ринку стартап-проекту

1	2	3
№ п/п	Найменування стану ринку	Характеристика
1	Кількість конкурентів	Майже відсутні
2	Загальний обсяг продаж, грн/од	100 000
3	Динаміка ринку	зростаюча
4	Специфічні вимоги до стандартизації	відсутні
5	Середня норма рентабельності в галузі, %	76

Висновки: головні переваги у даному стані ринку такі, що є стрімко та безперервно зростаюча динаміка попиту на даний вид технологій, що є підґрунтям для зростання кількості продажів. З загроз на ринку є високий необхідний рівень кваліфікації працівників. Кількість конкурентів мінімальна, адже всі відомі системи використовують технології, написані на інших мовах програмування, що дає великий простір для розвитку та захоплення ринку.

Великий середній рівень рентабельності означає, що проект має потенціал для входження на ринок.

Щоб отримати максимально чітку ситуацію на ринку та для подальшого аналізу проведемо визначення потенційних груп клієнтів (таблиця 4.4), їх характеристики, створимо перелік вимог цих груп до товару.

Таблиця 4.4 Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба	Потенційні клієнти	Відмінності потенційних клієнтів	Вимоги до товару
1	Системи збору потокових даних розроблені на мові Python	*Цивільне населення, малий, середній та великий бізнес, оборонна промисловість:* • агро бізнес • спорт • військова сфера • медицина • сфера обслуговування	Необхідність постійно масштабувати систему під обсяг та типи даних, місця та види їх зберігання, обробка даних	Відмовостійкість, Висока пропускна здатність, точність роботи, безпека даних

Висновки: провівши аналіз таблиці, бачимо формуючу потребу ринку. Це повна відсутність аналогічних рішень, які використовують дану мову програмування, хоча, важливо зазначити, про величезну кількість аналогів на інших мовах програмування, що може привести до необхідності обґрунтовувати вибір споживачами саме даного продукту.

Проведемо аналіз показників, що сприяють впровадженню даного проекту на ринок, та, навпаки, його занепаду.

4.3 Розроблення ринкової стратегії проекту

Сам процес створення ринкової стратегії для стартап-проекту потрібно починати з опису цільових клієнтів даного проекту(таблиця 4.5).

Таблиця 4.5 Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Фізичні особи	+-	+	Низька	+-
2	Юридичні особи	+	+	висока	+-
Яку цільову групу було обрано: при виборі цільової групи враховувалися наступні міркування. По-перше, враховувалася готовність споживачів прийняти продукт. Фізичні особи, тобто звичайні люди - необхідний більш загальний опис можливостей використання такої системи. Якщо реклама такої системи буде успішною, то з'явиться можливість заповнити нішу. Насамперед, продукт зацікавить юридичних осіб, тобто великих компаній та корпорацій, оскільки їм не потрібні додаткові пояснення.					

Висновки: За результатами оцінки було виокремлено групи, які потребують вжиття оперативних заходів. Відповідно, потрібна оперативна оцінка даних для подальшої реалізації конкретних заходів. Таблиця 4.5 показує, що обрані клієнти потребують цих рішень, але все ще не хочуть витрачати гроші, тому що не розуміють, навіщо їм такі рішення. Таблиця 4.5 показує, що обрані клієнти потребують цих рішень, але, як і раніше, неохоче витрачають на них гроші, оскільки не розуміють, навіщо вони їм потрібні.

Визначимо необхідну стратегію розвитку стартап-проекту (табл. 4.6).

Таблиця 4.6 Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку

	проекту		відповідно до обраної альтернативи	
1	Індивідуалізм	Стратегія недиференційованого маркетингу	Адаптація до вимог ринку Використання новацій Генерування ноу-хау	Стратегія спеціалізації

Далі необхідно вибрати стратегію конкуретної поведінки (табл. 4.7).

Таблиця 4.7 Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Стратегія конкурентної поведінки
1	Проект є першопрохідцем	Шукати нових споживачів із поступовим вивченням конкурентів	Стратегія виклику лідера

Висновки: проект є першопрохідцем, хоча має велику кількість конкурентів, хоча вони використовують інші технології для досягнення тих самих цілей. Проект має переваги над конкурентами, та оскільки може запропонувати просту та дешевшу альтернативу рішенням конкурентів, є можливо обрання стратегії лідера.

На основі вимог, які споживачі в обраних сегментах висувають до постачальників і продуктів, і відповідно до обраної базової стратегії розвитку та продуктів, визначаються вимоги до постачальників і продуктів.

Відповідно до обраної базової стратегії розвитку і стратегії конкурентної поведінки також розробляють стратегію позиціонування.

На основі стратегії конкурентної поведінки розробляється стратегія позиціонування (табл. 4.8).

Таблиця 4.8 Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Швидкість роботи	Стратегія спеціалізації	Дослідження та розвиток	Висока швидкість, надійність, відмовостійкість
2	Висока стабільність	Стратегія спеціалізації	Якість	Високоточна

Висновки: Ця стратегія важлива для подальшого розвитку проєкту й отримання прибутку для реінвестування в проєкт і нових закупівель.

4.4 Розроблення маркетингової програми та бізнес-моделі стартап-проєкту

Маркетингова концепція товару є першим кроком до формування ідеї яку отримує споживач, наведено у таблиці 4.9.

Таблиця 4.9 Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Необхідність відмовостойкої та простої системи	Проста у використанні система, яка розрахована на велике навантаження	Ціна, надійність, простота, підтримка, обслуговування, універсальність
2	Простота	Достатній функціонал,	Ціна, надійність,

	впровадження у інші системи, написані на мові Python	гнучка та ефективна система	простота, підтримка, обслуговування, універсальність

Висновки: в результаті оцінки маркетингової концепції є можливість створити рекламні кампанії для цільової аудиторії.

Розробимо трирівневу маркетингову модель товару (табл. 4.10).

Таблиця 4.10 Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Засіб дає можливість користувачеві значно розширити зону використання приладу при цьому зберігаючи високу точність орієнтації та керування		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	Доступність Швидкість роботи Точність Надійність Вартість	онлайн висока висока висока 11	Тх Тх Тх Тх Е
	Якість: сертифікати відповідності якості ISO, сертифікація для використання в людних місцях, сертифікація IEEE. Пакування: - Марка: -		
III. Товар із підкріпленням	Додавання нового функціоналу для користувача		
Захист товару проводиться шляхом постійного вдосконалення алгоритму та збільшенням кількості підтримуваних пристроїв.			

Висновки: Основним способом захисту програмного забезпечення є постійне вдосконалення алгоритмів і збільшення кількості сумісних пристроїв. Інший спосіб - постійно аналізувати та купувати найновіші розробки. Це означає придбання та впровадження новітніх розробок у даній сфері для підтримки актуальності системи.

Наступним кроком буде визначено цінові межі (табл. 4.11).

Таблиця 4.11 Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	300 у.о.	500 у.о.	1500 у.о.	300-450 у.о.

Висновки: Планується створити систему підписки на цей продукт, щоб забезпечити більш гнучку систему для користувачів, які бажають платити за більшу підтримку програмного забезпечення.

Визначимо оптимальну систему збуту, в межах якого будуть прийматися ті чи інші рішення(табл. 4.12).

Таблиця 4.12 Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Покупка товару, наявна функція підписки	Покупка товару для отримання розробленого ПО, наявна можливість створення системи підписки для доступу до оновлень алгоритму	Нульового рівня	Безпосередній (прямий)

Висновки: Ефективною буде вважаєтись система, яка по попередньому продажу товару та подальшому оформлені підписки допоможе отримувати кошти на постійній основі.

Таблиця 4.13 Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Обставини Змушують клієнтів шукати швидкі оцінки сигналу для зменшення витрат	Реклама SMM	Висока швидкість роботи Легкість Надійність Зручність Функціональність	Показати якість та надійність отримуваних результатів в ході експлуатації приладу з розробленим ПО	Показ зручності та можливості роботи з отриманими даними

Висновки: Маркетинг результатів здійснюється за допомогою різних рекламних каналів, включаючи рекламу в Facebook, Google, TikTok, LinkedIn, YouTube, Instagram. Цільові аудиторії повинні бути впевнені в якості та надійності результатів, які вони отримують при експлуатації своїх пристроїв з розробленим програмним забезпеченням.

Основними пунктами бізнес-моделі у стартап-проекті є: ключові партнери, ключова діяльність, ключові ресурси, ціннісна пропозиція, відносини з клієнтами, канали, сегменти користувачів, структура витрат, джерела доходів.

Таблиця 4.14 Бізнес-модель стартап-проекту

<table>
<tr>
<td rowspan="2">Ключові партнери. Компанії, що надають послуги з зберігання даних та використання їх серверних потужностей для їх обробки. У процесі дослідження та виробництва мінімально життєздатного продукту визначаються ключові постачальники. Основні ресурси, отримані від партнерів, - це їхні виробничі потужності та,</td>
<td>Ключові види діяльності. Розробка та підтримка життєздатності проекту. Вирішення проблем користувачів, якщо такі виникають, встановлення з ними зв’язку.</td>
<td rowspan="2">Ціннісна пропозиція. Продукт вирішує проблему потокової обробки даних для мови програмування Python, дозволяючи якісно та ефективно використовувати потокову обробку у проектах, що вже написані на цій мові, не переписуючи під іншу мову. Що дуже важливо, проект можна легко адаптувати та масштабувати під потреби</td>
<td>Відносини з клієнтами. Будуються після того, як продукт буде виведений на ринок. Можливо отримувати як персональну підтримку, так і самостійну роботу з виправлення проблем шляхом вивчення документації та інструкцій з використання програмного забезпечення.</td>
<td rowspan="2">Сегменти користувачів. Головною цільовою групою є компанії у різних сферах, що потребують програмне забезпечення з потокової обробки даних саме на мові Python.</td>
</tr>
<tr>
<td>Ключові ресурси. Інтелектуальн</td>
<td>Канали збуту. Контаки з</td>
</tr>
</table>

за необхідності, фінансування стартап-проекту.	і (програма, обладнання, дані, патент). Зв’язок з користувачам и або покупцями через джерела реклами або сайт.	клієнтів.	клієнтами відбувається у онлайн режимі або за допомогою зв’язку в телефонному режимі. Після мастабування та відкриття головного офісу можливо живе контактуванн я з замовниками.	
Структура витрат. Найбільші витрати проекту полягають у розробці та підтримці програмного забезпечення, а саме витрати на заробітну плату фахівцям та покупка серверного обладнання для зберігання даних.			**Джерела доходів.** Клієнти розраховують отримати якісний, простий та надійний продукт за свої кошти. Проект має фіксовану ціну з можливістю місячної підписки або разової покупки на визначені терміни.	

Висновки: проект є клієнтоорінтований з можливістю масштабування під потреби клієнта. Можливі різни види оплати (щомісячна підписка або повна оплата), що допомагає клієнтам зручно використовувати куплене програмне забезпечення.

Висновки до розділу

У цьому розділі описано стратегію ініціювання стартап-проекту. Метою є просування на ринок відмовостійкої та швидкодіючої системи по обробці потокових даних у всіх сферах життя де використовують Інтернет речей для бізнесу та персонального використання. Щоб підкреслити ідею проекту, описано інформацію в Таблиці 4.1. Ця інформація показує назву проекту, можливості, пропоновані замовнику при використанні програмного забезпечення, і ризики, на які наражається замовник. Також проаналізовано слабкі та сильні сторони та проведено порівняння з прямими конкурентами. Аналіз показав, що прямих конкурентів у цій галузі не багато, адже вони використовують інші мови програмування. Швидкість і надійність отриманих даних та швидка адаптація до різного обладнання також є сильними сторонами і важливими перевагами проекту. Технічний аудит проекту забезпечив краще розуміння найкращої технології для продукту. Результати показали, що технологія вже доступна.

Однак цього було недостатньо і потребувало вдосконалення. Загалом, проект можна було б розвивати у двох напрямках: розвиток можливостей для обробки ще більшої кількості потокових даних або впровадження більшої підтримки різних датчиків та серверного обладнання.

З огляду на стрімкий розвиток технологій, зокрема систем, що використовують та обробляють велику кількість даних, рівень розробки програмного забезпечення залишається відносно низьким, а тому необхідно приділяти більше уваги його вдосконаленню.

Розроблена бізнес-модель стартап-проекту демонструє вигідність інвестування в проект завдяки нішевому характеру проекту та відсутності великої кількості прямих конкурентів, які можуть запропонувати якісний продукт за ціною нижче середньоринкової.

ВИСНОВКИ

У результаті було розглянуто принципи побудови потокових систем збору даних, їх недоліки та переваги, важливі особливості та можливості використання, програмне забезпечення та технології, які використуються у даних системах.

Провівши аналіз особливостей систем, які використовують потокову обробку даних та написані на мові програмування Python була розроблена система збору даних на основі бібліотеки Quix Streams з використанням датчику MPU-6050 та платформи ESP8266.

Було роглянуто способи використання, внутрішня будова та особливості модулів та датчиків, які використані у розробці системи, розроблено схеми підключення цих датчиків. Також було розроблено програмне забезпечення для потокових систем збору експериментальних даних та розглянуто особливості його використання.

Використання даного програмного забезпечення дозволяє з мінімальними витратами ресурсів обробляти велику кількість інформації в режимі, що наближений до реального часу. Виходячи із висновків, зроблених при описі основної ідеї стартап-проекту, стало зрозуміло, що використання даної системи є досить простим, сама вона може бути прибутковою та більш ефективною у порівнянні з конкурентами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Наумовський А.А., Богомазов С.А. Програмне забезпечення для потокових систем збору експериментальних даних // Збірник праць XVI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “Погляд у майбутнє приладобудування”, 16-17 травня 2023р. – К.: ПБФ, КПІ ім. Ігоря Сікорського. – 2023. – С.223-225.
2. What is Quix? [Online]. Available: https://quix.io/docs/platform/intro.html. Accessed on: June 1, 2022.
3. Shapira, C., Palino, T., Sivaram, R., & Petty, K. (2022). *Kafka: The Definitive Guide.* O'Reilly Media, Inc.
4. Real time stream processing with Python [Online]. Available: https://quix.io/blog/stream-processing-framework-for-python/?returnUrl=https://quix.io/blog/p8/. Accessed on: November 27, 2022.
5. MPU6050 Module Pinout, Configuration, Features, Arduino Interfacing & Datasheet [Online]. Available: components101.com.
6. NodeMCU ESP8266 Pinout, Specifications, Features & Datasheet [Online]. Available: components101.com.
7. Гавриш, О. А., Бояринова К. О., Копішинська К. О. Розробка стартап-проектів. Конспект лекцій : навчальний посібник для студентів спеціальностей 151 – «Автоматизація та комп’ютерно-інтегровані технології» та 152 – «Метрологія та інформаційно-вимірювальна техніка»; КПІ ім. Ігоря Сікорського. Електронні текстові данні (1 файл: 2,88 Мбайт). Київ : КПІ ім. Ігоря Сікорського, 2019. 188 с. URL: http://ela.kpi.ua/handle/123456789/29447
8. Гавриш, О. А., Бояринова К. О., Копішинська К. О. Розробка стартап-проектів: практикум: навчальний посібник для студентів спеціальностей 151 – «Автоматизація та комп’ютерно-інтегровані технології» та 152 – «Метрологія та інформаційно-вимірювальна техніка»; КПІ ім. Ігоря

Сікорського. Електронні текстові данні (1 файл: 2,11 Мбайт). Київ : КПІ ім. Ігоря Сікорського, 2019. 116 с. URL: http://ela.kpi.ua/handle/123456789/29450

9. Покропивний С. Ф., Соболь С. М.,. Швиданенко Г. О,. Дерев'янко О. Г. Бізнес-план: технологія розробки та обґрунтування: навчальний посібник. Київ: КНЕУ, 2002. 379 с

ДОДАТОК А. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ЗБОРУ ПОТОКОВИХ ДАНИХ

А.1. Програмне забезпечення системи збору потокових даних

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;

void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10);

  Serial.println("Adafruit MPU6050 test!");

  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  Serial.print("Accelerometer range set to: ");
  switch (mpu.getAccelerometerRange()) {
  case MPU6050_RANGE_2_G:
    Serial.println("+-2G");
    break;
  case MPU6050_RANGE_4_G:
    Serial.println("+-4G");
    break;
  case MPU6050_RANGE_8_G:
    Serial.println("+-8G");
    break;
  case MPU6050_RANGE_16_G:
    Serial.println("+-16G");
    break;
  }
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  Serial.print("Gyro range set to: ");
```

```
switch (mpu.getGyroRange()) {
  case MPU6050_RANGE_250_DEG:
    Serial.println("+- 250 deg/s");
    break;
  case MPU6050_RANGE_500_DEG:
    Serial.println("+- 500 deg/s");
    break;
  case MPU6050_RANGE_1000_DEG:
    Serial.println("+- 1000 deg/s");
    break;
  case MPU6050_RANGE_2000_DEG:
    Serial.println("+- 2000 deg/s");
    break;
  }

  mpu.setFilterBandwidth(MPU6050_BAND_5_HZ);
  Serial.print("Filter bandwidth set to: ");
  switch (mpu.getFilterBandwidth()) {
  case MPU6050_BAND_260_HZ:
    Serial.println("260 Hz");
    break;
  case MPU6050_BAND_184_HZ:
    Serial.println("184 Hz");
    break;
  case MPU6050_BAND_94_HZ:
    Serial.println("94 Hz");
    break;
  case MPU6050_BAND_44_HZ:
    Serial.println("44 Hz");
    break;
  case MPU6050_BAND_21_HZ:
    Serial.println("21 Hz");
    break;
  case MPU6050_BAND_10_HZ:
    Serial.println("10 Hz");
    break;
  case MPU6050_BAND_5_HZ:
    Serial.println("5 Hz");
    break;
  }

  Serial.println("");
  delay(100);
}
```

```
void loop() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  Serial.print("Acceleration X: ");
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");

  Serial.print("Rotation X: ");
  Serial.print(g.gyro.x);
  Serial.print(", Y: ");
  Serial.print(g.gyro.y);
  Serial.print(", Z: ");
  Serial.print(g.gyro.z);
  Serial.println(" rad/s");

  Serial.print("Temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degC");

  Serial.println("");
  delay(500);
}
```

ДОДАТОК Б. СЕРВЕРНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

Б.1 Код серверного програмного забезпечення побудованого на базі бібліотеки Quix Streams.

Б.1.1 Файл producer.py

```
import quixstreams as qx
import time
import datetime
import math


client = qx.KafkaStreamingClient('127.0.0.1:9092')

topic_producer = client.get_topic_producer("quickstart-topic")

stream = topic_producer.create_stream()
stream.properties.name = "stream_data"
stream.properties.metadata["my-metadata"] = "my-metadata-value"
stream.timeseries.buffer.time_span_in_milliseconds = 100   # Send
data in 100 ms chunks


for index in range(0, 3000):
    stream.timeseries \
        .buffer \
        .add_timestamp(datetime.datetime.utcnow()) \
        .add_value("Acceleration", math.sin(index / 200.0)) \
        .add_value("Rotation", math.sin(index / 200.0)) \
        .add_value("Temperature", bytearray.fromhex("51 55 49
58")) \
        .publish()
    time.sleep(0.01)

print("Closing stream")
stream.close()
```

Б.1.2 Файл consumer.py

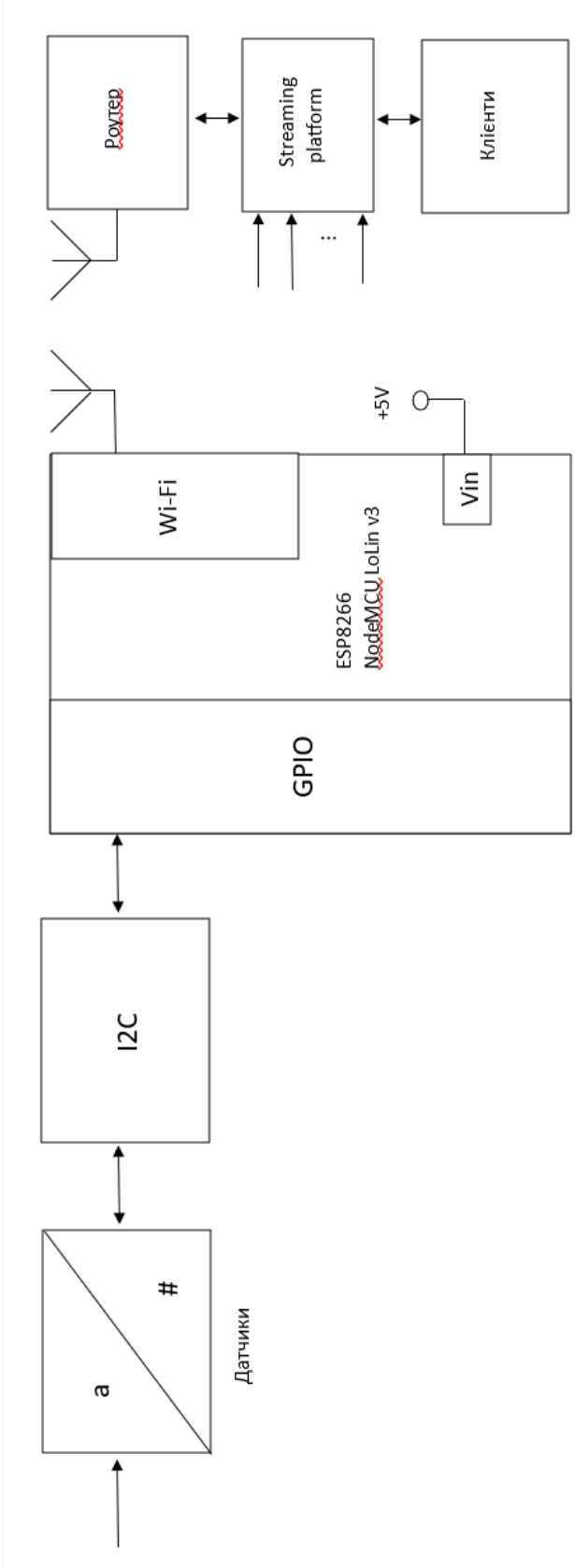
```
import quixstreams as qx
import pandas as pd

client = qx.KafkaStreamingClient('127.0.0.1:9092')
topic_consumer = client.get_topic_consumer("quickstart-topic",
consumer_group=None, auto_offset_reset=qx.AutoOffsetReset.Latest)
stream = topic_producer.create_stream()
stream.properties.name = "Stream data from MPU-6050"
```

```
def on_stream_received_handler(stream_consumer:
qx.StreamConsumer):
    stream_consumer.timeseries.on_data_received = callback_handler


    topic_consumer.on_stream_received = on_stream_received_handler
    topic_consumer.on_committed = g_running_total_per_stream.flush

    topic_consumer.on_stream_received = on_stream_received_handler
    topic_consumer.subscribe()

    stream_consumer.timeseries.on_dataframe_received =
on_dataframe_received_handler
   topic_consumer.on_stream_received = on_stream_received_handler

    commit_settings = CommitOptions()
    commit_settings.commit_every = 100
    commit_settings.commit_interval = 500
    commit_settings.auto_commit_enabled = True
    topic_consumer = client.get_topic_consumer(`stream_data`,
commit_settings=commit_settings)

    with (topic_producer := client.get_topic_producer(TOPIC_ID)):
    stream = topic_producer.create_stream()

    for index in range(0, 3000):
        stream.timeseries \
            .buffer \
            .add_timestamp(datetime.datetime.utcnow()) \
            .add_value("ParameterA", index) \
            .publish()
        time.sleep(0.01)
    print("Closing stream")
    stream.close()


with (producer := client.get_raw_topic_producer(TOPIC_ID)):
    data = bytearray(bytes("TEXT CONVERTED TO BYTES", 'utf-8'))

    message = RawMessage(data)
    message.key = MESSAGE_KEY
    producer.publish(message)

g_running_total_per_stream =
qx.InMemoryStorage(qx.LocalFileStorage())


def callback_handler(stream_consumer: qx.StreamConsumer, data:
qx.TimeseriesData):
    if stream_consumer.stream_id not in
g_running_total_per_stream:
```

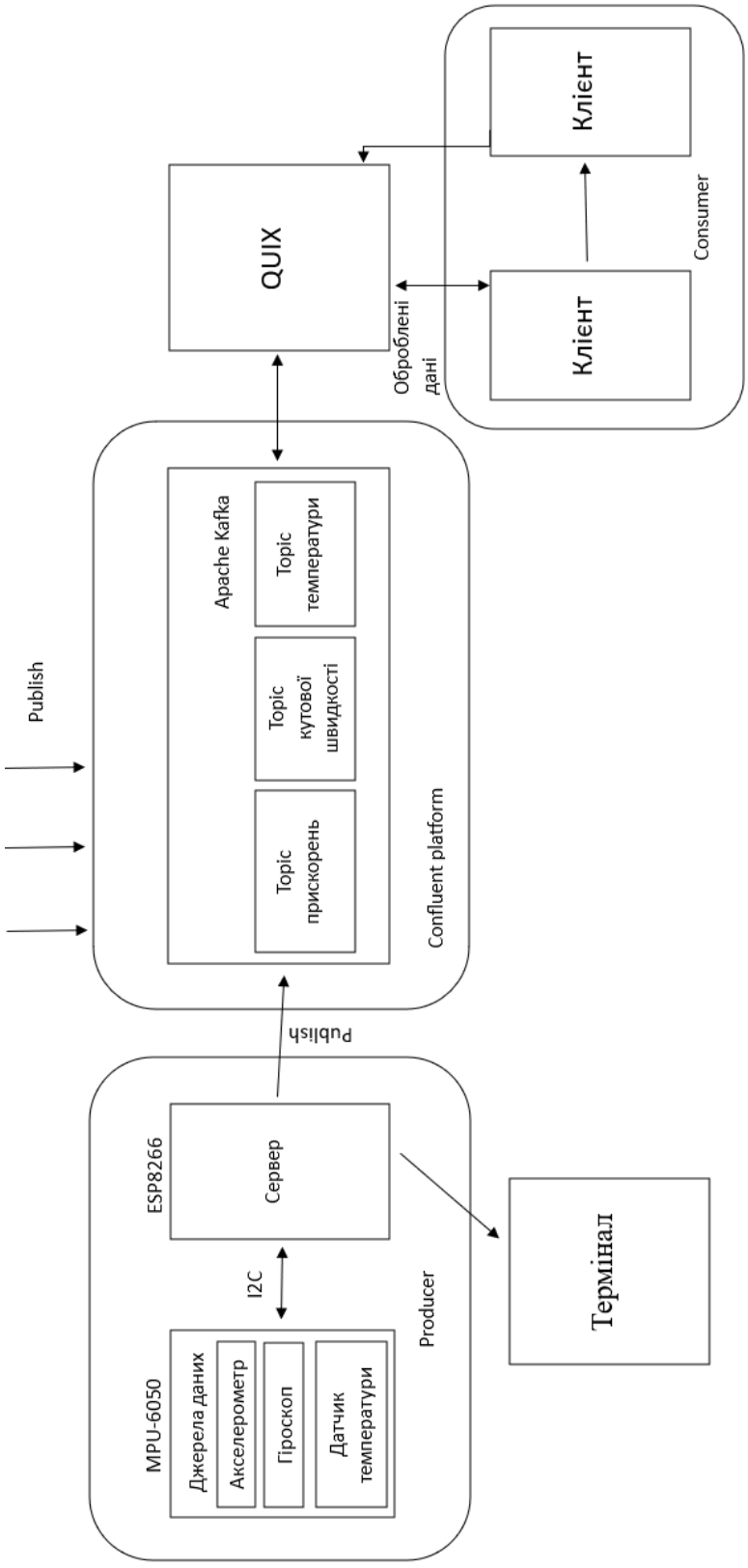
```
        g_running_total_per_stream[stream_consumer.stream_id] = 0


g_running_total_per_stream[stream_consumer.stream_id] +=
some_value


def on_dataframe_received_handler(stream: qx.StreamConsumer, df:
pd.DataFrame):
    print(df.to_string())

topic_consumer.on_stream_received = on_stream_received_handler

def on_stream_closed_handler(stream: StreamConsumer, end_type:
StreamEndType):
    print("Stream closed with {}".format(end_type))


print("Listening to streams. Press CTRL-C to exit.")
qx.App.run()
```

ДОДАТОК В. ГРАФІЧНІ МАТЕРІАЛИ

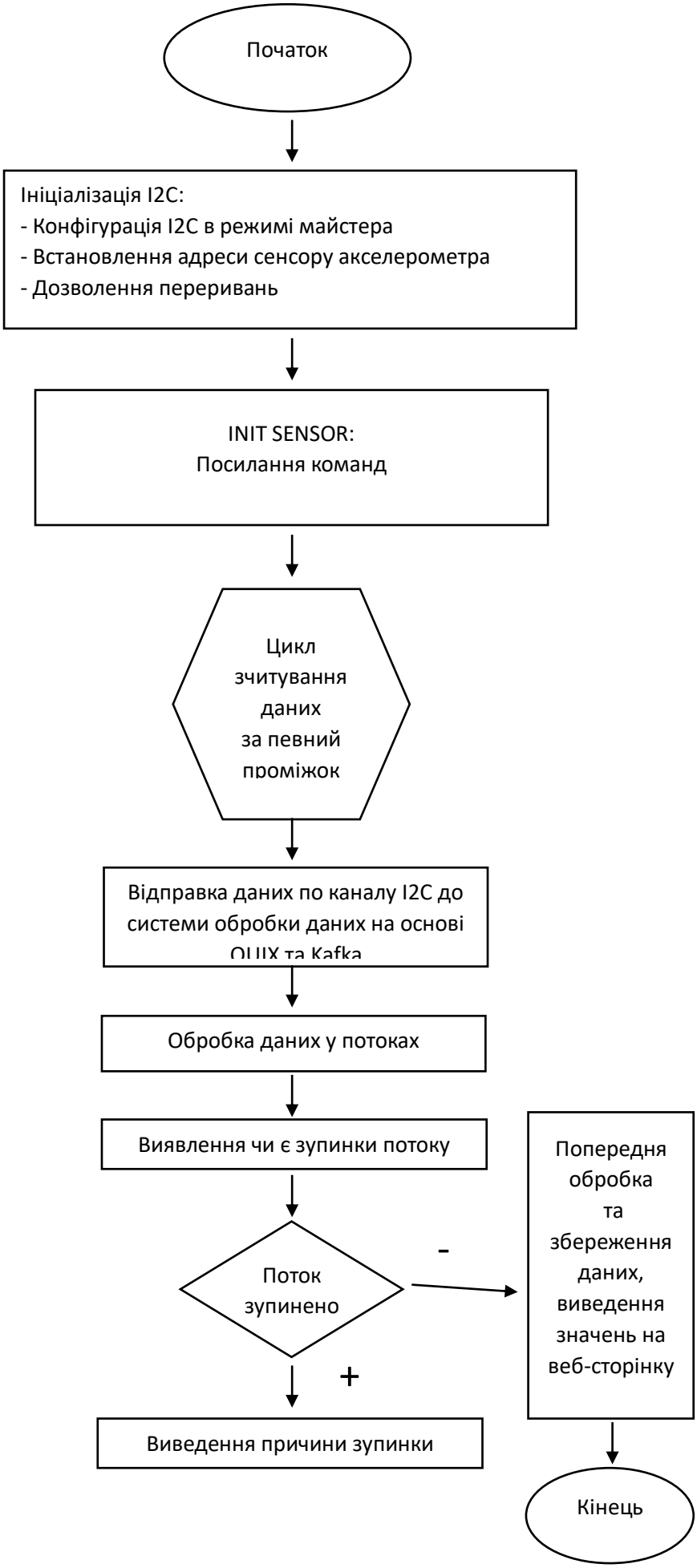
В.1 Структура апаратно-програмного забезпечення системи збору експериментальних даних на основі потокових технологій



В.2 Функційна схема апаратно-програмного забезпечення



В.3 Блок схема роботи апаратно-програмного забезпечення



ДОДАТОК Г. СПИСОК ПУБЛІКАЦІЙ

Наумовський А.А., Богомазов С.А. Програмне забезпечення для потокових систем збору експериментальних даних // Збірник праць XVI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “Погляд у майбутнє приладобудування”, 16-17 травня 2023р. – К.: ПБФ, КПІ ім. Ігоря Сікорського. – 2023. – С.223-225.

УДК 621.3.087

А.А. Наумовський, студент гр. ПІ-21мп, к.т.н., доц. Богомазов С.А.
КПІ ім. Ігоря Сікорського

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОТОКОВИХ СИСТЕМ ЗБОРУ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

Анотація. В даній статті було розглянуто особливості розробки програмного забезпечення на основі бібліотеки Quix для мови програмування Python, яка використовується для потокової передачі телеметричних даних. Розглянуто можливості бібліотеки та наведено приклади її використання. Були розглянуті принципи потокової обробки даних, проблеми, що виникають в процесі передачі даних та шляхи їх вирішення. Розроблене програмне забезпечення може бути використане в багатьох галузях промисловості та різних сферах діяльності для обробки великої кількості даних в режимі реального часу.

Ключові слова: потокова передача даних, потік даних, мікросервіси, Quix, Python, Kafka, програмування, промисловість.

ВСТУП

Коректна та своєчасна обробка великої кількості даних - дуже важливий та складний процес, адже будь-який втрачений або несвоєчасно опрацьований байт даних в багатьох випадках може привести до аварійної ситуації.

Традиційна архітектура програмного забезпечення для обробки даних в більшості випадків є орієнтованою на бази даних. Це означає, що для обробки даних необхідно неодноразово звернутися до бази даних (рис.1).

Рисунок 1. Схема обробки даних на основі використання бази даних

Але в умовах обробки, що наближена до реального часу, потрібно отримувати результати в масштабі секунд або наносекунд. Наприклад, коли мільйони пристроїв IoT одночасно надсилають дані для обробки, традиційні архітектури, орієнтовані на бази даних, не задовольняють вимогам реального часу [1].

Потокова обробка використовується для отримання безперервного потоку даних та швидкої обробки подій в цьому потоці за невеликий проміжок часу після отримання даних. Час обробки може варіюватися від мілісекунд до хвилин. В багатьох випадках дані швидко деградують, тому за час, необхідний системі для збору, зберігання, обробки та надання даних, інформація часто змінюється. За допомогою штучного інтелекту, машинного навчання та обробки потоків у реальному часі можна використовувати дані способами, які інакше були б неможливими. Потокова обробка надає можливості реагувати в режимі реального часу і оперативно вносити потрібні зміни.

◢ Для розробників які працюють з мовою програмування Python, реалізація програмного забезпечення на основі потокових технологій обробки даних є досить ускладненою. Це обумовлено тим, що ця мова програмування не сумісна з бібліотеками, що використовуються для реалізації потокових технологій (такими як Kafka), які переважно розроблені на інших мовах програмування, наприклад, Java та Scala. Платформа Kafka Streams була розроблена на мові програмування Java для забезпечення максимальної продуктивності. Мова програмування Python в ній підтримується для досить обмеженого кола задач, таких як публікація, підписка на повідомлення від брокера та виконання простих операцій [2]. Багато програмних платформ, які підтримують потокову обробку даних, не сумісні з мовою Python і розробникам доводиться переписувати код на інших мовах програмування.

МАТЕРІАЛИ І РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Використовувати переваги потокової обробки даних на основі технологій Python дозволяє бібліотека Quix Streams [3]. Це бібліотека з відкритим кодом для потокової передачі та обробки телеметричних даних. Вона призначена для використання в службах високочастотної телеметрії, коли необхідно обробляти великі обсяги даних часових рядів з точністю до наносекунд. Розроблена для Python клієнтська бібліотека Quix Python дозволяє фахівцям з обробки та аналізу даних обробляти потокові дані та розгортати моделі машинного навчання та інформаційні продукти без втручання розробників та інших технічних фахівців. Quix Streams використовує брокер повідомлень Apache Kafka замість бази даних, що дозволяє обробляти дані часових рядів з високою продуктивністю.

Бібліотека надає користувачу веб-інтерфейс, можливість використання онлайн IDE, зразки коду, готові до запуску та розгортання, швидке і просте розгортання потокової передачі та керування через інтерфейс користувача, інструменти моніторингу для відстеження стану і даних потокових програм у режимі реального часу, зручну візуалізацію у вигляді конвеєра та провідник даних для перевірки даних, оброблених програмою. Програмне забезпечення має чотири види API (Application Programming Interface): Stream Writer API призначене для запису даних у систему обміну повідомленнями між серверними додатками Kafka за допомогою HTTP і містить методи шифрування, серіалізації та перетворення даних у формат Quix Streams незалежно від джерела даних; Stream Reader API відправляє дані у програму, що буде обробляти ці дані та забезпечує мінімальну затримку передачі даних, уникаючи будь-яких операцій з диском; Data Catalogue API містить записи попередніх потоків даних та дозволяє створювати інформаційні панелі та експортувати дані в інші системи; Portal API для автоматизації процесу передачі потоків даних та створення робочих областей.

Quix використовує брокер повідомлень і робить його центром програмного забезпечення, виконуючи обробку даних без необхідності зберігати та передавати інформацію через базу даних (рис.2).

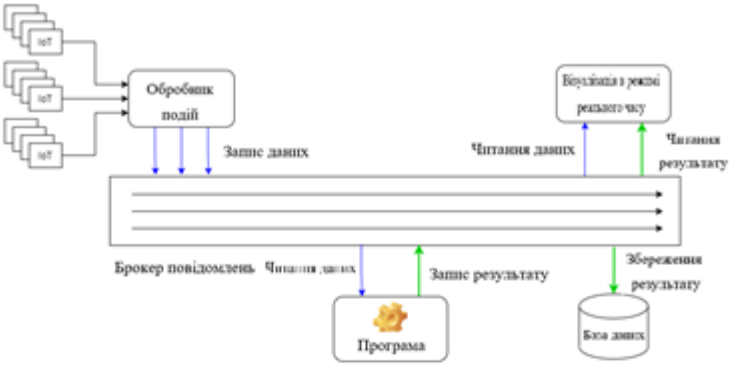


Рисунок 2. Обробка даних за допомогою бібліотеки Quix

Використовуючи обробку в пам'яті програма може зберігати лише ті дані, які дійсно необхідно використати. Такий підхід знижує складність і вартість обробки даних в реальному часі на кілька порядків і дозволяє обробляти велику кількість даних при низьких затримках (рис.3).

Рисунок 3. Відображення даних від датчиків в режимі реального часу

ВИСНОВКИ

У результаті було розроблено програмне забезпечення на мові програмування Python з використанням бібліотеки Quix для потокових систем збору експериментальних даних та розглянуто особливості його використання. Використання даного програмного забезпечення дозволяє з мінімальними витратами ресурсів обробляти велику кількість інформації в режимі, що наближений до реального часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] What is Quix? [Online]. Available: https://quix.io/docs/platform/intro.html. Accessed on: June 1, 2022.

[2] Shapira, C., Palino, T., Sivaram, R., & Petty, K. (2022). *Kafka: The Definitive Guide*. O'Reilly Media, Inc.

[3] Real time stream processing with Python [Online]. Available: https://quix.io/blog/stream-processing-framework-for-python/?returnUrl=https://quix.io/blog/p8/. Accessed on: November 27, 2022.