МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломної роботи

магістра
(освітній ступінь)

на тему «Розробка програмно-апаратної системи вібродіагностики промислового обладнання. Клієнтська частина»

ХАІ.603.667П2.121.166381.20О

Виконав: студент 6 курсу групи №667П2
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Лезновський О.А
(прізвище й ініціали студента)
Керівник: Сергієнко В.В.
(прізвище й ініціали)
Рецензент: Мартовицький В.О.
(прізвище й ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____ Туркін І.Б._____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

_____Лезновському Олексію Андрійовичу_____
(прізвище, ім’я, по батькові)

1. Тема дипломної роботи Розробка програмно-апаратної системи вібродіагностики промислового обладнання. Клієнтська частина
Керівник дипломної роботи Сергієнко Володимир Володимирович к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету № _____ від “____” _____ 2020 року
2. Термін подання студентом роботи ___20.11.2020_____
3. Вихідні дані до роботи виконати аналіз проблем розробки програмно-апаратної системи вібродіагностики промислового обладнання
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)___розробка програмно-апаратної системи вібродіагностики промислового обладнання.
5. Перелік графічного матеріалу 2 слайди, аналіз та постановка задачі – 2 слайди, моделі і методи проведення вібраційної діагностики – 3 слайди, розроблення прототипу програмного забезпечення системи вібродіагностики – 5 слайдів, Перевірка працездатності розроблених моделей та методів – 2 слайди, висновки – 1 слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1÷3	Сергієнко В.В. доцент каф.603		

Нормоконтроль _____ __Постернакова В.А_ «___»_____20__р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання «___»_____20_____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд і аналіз рішень для проведення вібродіагностики обладнання	01.09.2020	
2	Аналіз методів проектування вбудованого програмного забеспечення	12.09.2020	
3	Аналіз методів реалізації мережевого обміну	15.09.2020	
4	Розробка методу і алгоритмів вібродіагностики обладнання	20.09.2020	
5	Розробка прототипу ПЗ	01.10.2020	
6	Оформлювання пояснювальної записки до дипломного проекту	20.11.2020	
7	Підготовка доповіді	22.11.2020	
8	Підготовка презентації	30.11.2020	
9	Захист дипломного проекту	14.12.2020	

Студент_____Лезновський О.А
(підпис) (прізвище та ініціали)

Керівник проекту __ ____Сергієнко В.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Записка на дипломну роботу: 58 с., 57 іл., 10 джерел.

Тема роботи - Розробка програмно-апаратної системи вібродіагностики промислового обладнання. Клієнтська частина.

Мета дипломної роботи магістра – підвищення ефективності експлуатації обладнання на промислових підприємствах шляхом використання бездротової системи вібродіагностики.

Для досягнення мети необхідно виконати наступні завдання:

- провести аналіз проблем вібраційного моніторингу обладнання;
- вибрати інструментальні засоби розробки програмного і апаратного забезпечення для проведення вібраційного діагностики;
- розробити архітектуру апаратної платформи;
- виконати проектування архітектури вбудованого програмного забезпечення;
- розробити прототип програмно-апаратної платформи і стенда для тестування;

Актуальність завдання - створення інформаційної системи вібраційної діагностики яка буде корисна для аналізу стану обладнання на підприємстві та підвищить безпеку експлуатації устаткування і знизить кількість простоїв обладнання.

Основним практичним результатом роботи програмного комплексу є розробка інструментальних засобів вібраційної діагностики а також стенду для верифікації розробленої системи.

Необхідно проаналізувати методи і моделі вібраційної діагностики а також розробити програмно-апаратну систему яка реалізує можливість оцінки вібраційного стану обладнання у реальному часі.

Основною метою впровадження програмного-апаратного комплексу є підвищення ефективності експлуатації обладнання на промислових підприємствах.

ВІБРОДИАГНОСТИКА, ІНФОРМАЦІЙНА СИСТЕМА, ПРОМИСЛОВИЙ ІНТЕРНЕТ РЕЧЕЙ, ОЦІНКА СТАНУ ОБЛАДНАННЯ, ПРОГРАМНО-АППАРАТНА ПЛАТФОРМА

РЕФЕРАТ

Записка на дипломну роботу: 58 с., 57 іл., 10 джерел.

Тема работы - разработка программно-аппаратной системы вибродиагностика промышленного оборудования. Клиентская часть.

Цель дипломной работы магистра - повышение эффективность эксплуатации оборудования на промышленных предприятиях путем использования беспроводной системы вибродиагностики.

Для достижения цели необходимо выполнить следующие задачи:

-провести анализ проблем вибрационного мониторинга оборудования;

-выбрать средства разработки программного и аппаратного обеспечения для проведения вибрационного диагностики;

-разработать архитектуру аппаратной платформы;

-выполнить проектирование архитектуры встроенный программного обеспечения;

-разработать прототип программно-аппаратной платформы и стенда для тестирования;

Актуальность задачи - создание информационной системы вибрационной диагностики, которая будет полезна для анализа состояния оборудования на предприятии, снизит количество простоев и повысит безопасность эксплуатации оборудования.

Основным практическим результатом работы программного комплекса является разработка инструментальных средств вибрационной диагностики, а также стенда для верификации разработанной системы.

Необходимо проанализировать методы и модели вибрационной диагностики, а также разработать программно-аппаратного систему, которая реализует возможность оценки вибрационного состояния оборудования в реальном времени.

Основной целью внедрения программного-аппаратного комплекса является повышение эффективность эксплуатации оборудования на промышленных предприятиях.

ВИБРОДИАГНОСТИКА, ИНФОРМАЦИОННАЯ СИСТЕМА, ПРОМЫШЛЕННЫЙ ИНТЕРНЕТ ВЕЩЕЙ, ОЦЕНКА СОСТОЯНИЯ ОБОРУДОВАНИЯ, ПРОГРАММНО-АППАРАТНАЯ ПЛАТФОРМА

ABSTRACT

Explanatory note for thesis: 58 p., 57 ill., 10 sources.

Work theme - development of a software and hardware system for vibrodiagnostics of industrial equipment. Client side.

The goal of the master's degree project is to increase the efficiency of equipment operation at industrial enterprises by introducing a wireless vibration diagnostics system.

To achieve the goal, you must complete the following tasks:

- to analyze the problems of vibration monitoring of equipment;
- analyze approaches to the implementation of software and hardware for vibration diagnostics;
- analyze the features of the hardware platform;
- to analyze the methods of processing the received data;
- to design the architecture of the embedded software;
- to develop a prototype of a software and hardware platform and a test bench;

The urgency of the task is to create an information system that will be useful for analyzing the state of equipment at the enterprise, as well as increase the safety of equipment operation and reduce the number of equipment downtime.

The main practical result of the software complex is the development of vibration diagnostics tools as well as a stand for verification of the developed system.

It is necessary to develop methods and models, which is the basis for the development of software and hardware systems for vibration diagnostics and the development of embedded software.

The main purpose of the implementation of the software and hardware complex is to increase the efficiency of equipment operation at industrial enterprises.

VIBRODIAGNOSTICS, INFORMATION SYSTEM, INDUSTRIAL INTERNET OF THINGS, EVALUATION OF EQUIPMENT, SOFTWARE AND HARDWARE SOLUTION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

БД – база даних;

ОС – операційна система;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

Пакет – в телекомунікаціях це відформатований блок даних який передається за допомогою комутації пакетів в комп'ютерній мережі;

СУБД – система управління базою даних;

Фреймворк – вузько направлена бібліотека для вирішення завдань;

IoT(Internet of things) – це мережа пов'язаних через інтернет об'єктів, здатних збирати дані і обмінюватися даними, які надходять зі вбудованих сервісів;

MVC(Model-View-Controller) – архітектурний шаблон «Модель–вигляд–контролер».

ВСТУП

На сьогоднішній день можливості з побудови автоматизованих систем з мікропроцесорним управлінням надзвичайно широкі. Раніше такі системи застосовувалися в промисловості та оборонних програмах, але з настанням 21 століття такі рішення стали доступні і для побутових застосувань.

Прикладом може служити ринок інтернету речей і автоматизації енергопостачання будівель. На сьогодні, побудова системи автоматичного контролю енергоспоживання є безальтернативною можливістю для підприємств завчасно формувати звітність.

Всі перераховані приклади можуть мати різноманітну конфігурацію, для кращої відповідності очікуванням цільової аудиторії. Один з перспективних ринків - ринок, орієнтований на продаж таких систем кінцевим побутовим споживачам. Такі системи можуть бути встановлені і налаштовані самим користувачем, без необхідності вдаватися до послуг висококваліфікованих фахівців.

Дана робота торкнеться актуальної теми інтеграції подібних пристроїв з єдиною інформаційною системою підприємства. Така інтеграція відрізняється від звичних багатьом сценаріїв взаємодії комп'ютерів з периферійними пристроями. В першу чергу це відмінність полягає в застосуванні у вбудованих системах іншого набору комутаційних портів і протоколів з'єднання. Так само відрізняються сценарії синхронізації і взаємодії персонального комп'ютера з такими пристроями.

Науково-прикладна задача - експериментально оцінити доцільність застосування методів, алгоритмів та технічних рішень для оцінки стану обладнання.

Метою роботи є підвищення ефективності роботи обертового обладнання за допомогою оперативної вібраційної діагностики з використанням MEMS акселерометрів. Для досягнення мети необхідно виконати наступне завдання – розробити програмно-апаратний комплекс вібраційної діагностики для оперативної оцінки стану обладнання.

Завдання для досягнення мети:

1) провести аналіз методів оцінки та обробки показників датчиків вібрації;
2) обрати варіанти технічних рішень діагностики обладнання;
3) виконати планування експерименту для дослідження системи вібраційної діагностики обладнання;
4) розробити алгоритми оброблення результатів експерименту для дослідження системи вібраційної діагностики обладнання;
5) на основі аналізу результатів експерименту сформулювати практичні рекомендації застосування розроблених методів та програмно технічних рішень для оцінки обладнання за допомогою вібраційної діагностики.

Об’єктом дослідження є оцінка стану обладнання за допомогою вібраційної діагностики.

Предметом дослідження є метод оцінки стану обладнання за допомогою аналізу вібраційних показників.

Методи дослідження:

1) аналіз – метод буде використовуватися для виділення основних ознак при класифікації дефектів які призводять до некоректної роботи обертового обладнання;
2) порівняння – метод буде використовуватися при визначенні класів пошкоджень та їх розмежуванні;
3) вимірювання – процедура визначення чисельного значення певної величини за допомогою одиниці виміру. Цей метод буде використовуватися для оцінювання точності розроблюваної системи;
4) експеримент – метод буде використовуватися при моделюванні;
5) матеріальне моделювання – метод що використовується при моделюванні роботи обертового обладнання за допомогою стенду.

Наукова новизна - удосконалений метод вібраційної діагностики для автоматизованого оцінювання стану обладнання. Запропоновані методи і варіанти реалізації дозволяють проводити комплексну оцінку стану обладнання в режимі реального часу

Практична значущість отриманих результатів полягає в тому що впровадження подібної системи дозволить користувачам зменшити витрати на проведення вібраційного діагностики, а також змінити модель обслуговування обладнання на «обслуговування за реальним станом обладнання».

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ ПРОВЕДЕННЯ ВІБРАЦІЙНОІ ДІАГНОСТИКИ ОБЛАДНАННЯ

Сучасні технологічні процеси потребують безперервного контролю за багатьма параметрами технологічного обладнання. Одними з найважливіших є механічні параметри, зокрема механічні вібрації досліджуваного об’єкта. Подібний контроль необхідний в різних областях науки та техніки. Наприклад, в напівпровідниковій електроніці – для контролю вібрації установок вирощування кристалів, а в мікроелектроніці – для контролю вібрацій установок фотолітографії. В машинобудуванні такий контроль використовується для визначення вібрацій верстатів і биття деталей та вібрацій ріжучого інструменту, а в автомобільній промисловості – щоб контролювати вібрації окремих вузлів автомобіля і всього автомобіля в цілому. На залізничному транспорті контролюють вібрації, щоб визначити наближення поїзда, в енергетиці – для контролю вібрації лопаток газових турбін та контролю вібрацій в газопроводах, а в авіабудуванні – щоб контролювати биття турбін і т.д.

Системи моніторингу вібрацій дають змогу вирішити численні проблеми пов’язані з вібраціями, які виникають при роботі різного технологічного обладнання (механізмів, машин, верстатів з ЧПК). Отже, розробка систем призначених для моніторингу та аналізу вібрацій технічних об’єктів є актуальним питанням сьогодення.

1.1 Аналіз літературних джерел та постановка проблеми вібраційної діагностики обладнання

Проведений аналіз існуючих літературних джерел дає змогу стверджувати, що існує два основні методи вимірювання параметрів вібрації: контактні, які мають механічний зв’язок давача з досліджуваним об’єктом, і безконтактні, тобто які не пов’язані з об’єктом механічним зв’язком.

Контактні методи є найпростішими методами вимірювання вібрації за допомогою п’єзоелектричних датчиків. Контактні методи дають можливість проводити вимірювання з високою точністю в діапазоні низьких частот і відносно великих амплітуд вібрації, але внаслідок своєї високої інерційності приводять до спотворення форми сигналу, що унеможливлює вимірювання вібрації високої частоти і малої амплітуди. Крім того, якщо маса досліджуваного об’єкта, а отже і його інерційність, невелика, то такий давач може істотно впливати на характер вібрації, що вносить додаткову похибку у вимірювання.

Ці недоліки усуває метод відкритого резонатора. Суть методу полягає у вимірюванні параметрів високочастотного резонатора, які змінюються внаслідок вібрації досліджуваного об’єкта. Резонатор має два дзеркала, причому одне з них фіксоване, а інше механічно пов’язане з досліджуваним об’єктом. Реєстрація переміщень при малих амплітудах вібрації проводиться амплітудним методом по зміні вихідної потужності. Цей метод вимірювання

вимагає сталої потужності, що підводиться до резонатора і високої стабільності частоти збудження.

Всі безконтактні методи вимірювання вібрації ґрунтуються на зондуванні об’єкта звуковими і електромагнітними хвилями. Однією з останніх розробок є метод ультразвукової фазометрії. Основна ідея якого полягає у вимірюванні поточного значення різниці фаз опорного сигналу ультразвукової частоти та сигналу, відбитого від досліджуваного об’єкта. В якості переваг методу можна відзначити дешевизну і компактність апаратури, малий час виміру, відсутність обмеження на частотний діапазон, високу точність вимірювання низькочастотних вібрацій. Недоліками є сильне згасання ультразвуку в повітрі, залежність від стану атмосфери, зменшення точності вимірювання із зростанням частоти вібрації.

Великого поширення набули методи, які ґрунтуються на зондуванні об’єкта видимим світлом. Всі оптичні методи поділяються на дві групи. До першої відносять методи, засновані на реєстрації ефекту Допплера . Найпростішим з них є гомодинний метод, який дає змогу вимірювати амплітуди і фази гармонійних вібрацій, але з його допомогою неможливо досліджувати негармонійні і великі за амплітудою вібрації. Ці недоліки можна усунути використовуючи гетеродинні методи. Але вимагають калібрування і, окрім того, складної вимірювальної апаратури. Суттєвим недоліком перерахованих вище методів є високі вимоги до якості поверхні досліджуваного об’єкта. Але втрачають своє значення при використанні голографічних методів, які утворюють другу групу. Голографічні методи мають високу роздільну здатність, однак вимагають складного і дорогого обладнання. Крім того, час вимірювань дуже великий. Загальними недоліками оптичних методів вимірювання вібрації є складність, громіздкість і висока вартість обладнання, велике енергоспоживання, високі вимоги до якості поверхні досліджуваного об’єкта, високі вимоги до стану атмосфери (певна вологість, відсутність запиленості і т. п.). Крім того, лазерне випромінювання має шкідливий вплив на зір обслуговуючого персоналу і вимагає додаткових запобіжних заходів і захисту.

Розробленню систем моніторингу та аналізу вібрацій присвячено різні науково-технічні статті. Зокрема, в [1] розроблено систему моніторингу та аналізу вібрацій, що виникають в електромоторах. Система використовує п’єзоелектричний акселерометр (ICP 603C11) і плату збору даних від National Instruments (NI 6009). Вібраційні сигнали збираються з різних частин електричних моторів і передаються на комп’ютер через плату збору даних. Віртуальний інструмент, що дає змогу в реальному часі моніторити і проводити Фур’є аналіз отриманих сигналів з сенсора вібрацій реалізовано в LabVIEW [3]. В [2] розроблено вбудовану систему для моніторингу вібрацій насосного агрегату на базі мікроконтролера від компанії Microchip. Програмне забезпечення (ПЗ) для збору і аналізу даних оптимізовано для тестування помп з турбонадувом з швидкостями обертання до 2000 об/хв. Обмеження ПЗ встановлено на автоматичну діагностику, але може бути налаштоване для індивідуальної і ручної вібродіагностики. Єдиним обмеженням системи є

характеристики акселерометра. Автори провели велику кількість вимірювань за допомогою розробленої системи на різних турбоагрегатах для визначення експлуатаційних умов насосних агрегатів [2]. В [3] описано метод визначення перемішення та швидкості з сигналів прискорення отриманих з акселерометрів, а в [4] розроблено методику моніторингу надійності мостових конструкцій використовуючи МЕМС акселерометри. В [4] побудовано систему моніторингу верстатів і процесів механічної обробки. Система збору вібраційного сигналу базується на мікроконтролері Arduino, який підключено до комп'ютера через USB порт. Спеціальне розроблене ПЗ під LabVIEW зчитує та опрацьовує дані в реальному масштабі часу. В [5] досліджена придатність мікро-електромеханічних (МЕМС) акселерометрів для моніторингу стану верстатів з ЧПК. Тести проведено на реально-діючому верстаті з ЧПК в типовому промисловому цеху. Показано, що МЕМС давачі можуть бути хорошою альтернативою до стандартних сенсорів вібрації, оскільки вони не потребують важких електрометричних підсилювачів. Вибір такого давача має бути зроблений відповідно до вимог застосування і результату тесту на придатність. МЕМС давачі можуть використовуватися у жорстких умовах використовуючи спеціальне упакування. Ряд авторів використовують мікроконтролери Arduino в апаратно-програмній системі для вимірювання механічних вібрацій [6]. В якості давачів вібрації використано акселерометри ADXL335. Розроблена система використовувалася для дослідження та моніторингу вібрацій вакуумної помпи. Дослідження показали можливість і доцільність розроблення вбудованих систем моніторингу вібрацій в реальному масштабі часу з використанням не дорогого апаратного та ПЗ. Використання іншого типу мікроконтролера наведено в роботі [7]. Зокрема, запропоновано систему моніторингу вібрацій ротаційних машин, верстатів, яка побудована на мікроконтролері PIC-18F6520 і акселерометрі ADXL322. Проведено дослідження на перевірку можливості реєструвати піки частот появи відмов для різних випадків несправностей. Найкращими підходами і технічними рішеннями серед описаних вище для розв'язання розглядуваного кола задач можна віднести методи, які описані в роботах [1], [3]. Методи дослідження грунтувалися на проведенні експериментів з різними технічними об'єктами, що включали вимірювання параметрів вібрації, їх обробку та аналіз за допомогою власно розроблених апаратно-програмних систем. Розроблені системи є закритими та мають високу ціну, що не дає змоги розширювати їх функціональні можливості та модифікацію до відповідних потреб експлуатації. Отже, аналіз існуючих підходів та технічних рішень привів до розроблення якісно нової недорогої відкритої апаратно-програмної системи моніторингу вібрацій в реальному масштабі часу. Така система має бути побудована на доступних і недорогих комплектуючих, відкритому ПЗ та з можливістю модифікації або розширення її функціональних можливостей відповідно до вимог та області застосування.

1.2 Аналіз особливостей використання наявних типів датчиків

Виходячи з аналізу, проведеного в розділі 1.1, найбільш підходящим варіантом є використання датчиків MEMS типу. Особливістю таких датчикі є малий розмір корпусу(напр. 5x5мм для ADXL345) та низька ціна. Більшість таких датчиків підтримують інтерфейс SPI. З найбільш сучасних моделей можливо виділити iis3dwb, який позиціонуються виробником як спеціалізоване рішення для розробки віброметрів промислового обладнання.

Для інтерфейсу SPI швидкість залежить від можливостей мікроконтролера та акселерометру. Максимальна частота тактового сигналу для МК складає 45MHz. Для акселерометру це значення складає 10 MHz.

Обмін інформацією з акселерометром відбувається за таким алгоритмом. Спочатку, МК відсилає по шині SPI адрес регістра пристрою з якого буде проходити зчитування або запис даних. Тоді, відсилаються дані для запису у цей регістр або зчитується визначена кількість байт з пристрою.

Ініціалізація акселерометра IIS3DWB складається з трьох кроків:

1) Ініціалізація акселерометра для роботи з перериваннями
2) Встановлення значення дискретизації
3) Встановлення максимального значення та розподільної здатності

Встановлення режиму переривать відбувається за допомогою регістру INT1_CTRL. Склад регистру та можливі значення продемонстровано на рисунку 1.1:

INT1_CTRL register

0(1)	INT1_ CNT_BDR	INT1_ FIFO _FULL	INT1_ FIFO_ OVR	INT1_ FIFO_TH	INT1_ BOOT	0(1)	INT1_ DRDY_XL

1. *This bit must be set to '0' for the correct operation of the device.*

INT1_CTRL register description

INT1_CNT_BDR	Enables COUNTER_BDR_IA interrupt on INT1.
INT1_ FIFO _FULL	Enables FIFO full flag interrupt on INT1 pin.
INT1_ FIFO_ OVR	Enables FIFO overrun interrupt on INT1 pin.
INT1_FIFO_TH	Enables FIFO threshold interrupt on INT1 pin.
INT1_BOOT	Enables boot status on INT1 pin
INT1_ DRDY_XL	Enables accelerometer data-ready interrupt on INT1 pin.

Рисунок 1.1 - Склад регістру та можливі значення

Для обраного сценарію використання необхідно встановити біт INT1_DRY_XL у 1.

Наступним кроком буде встановлення значення дискретізації за допомогою регістру INTERNAL_FREQ_FINE. Склад регістру продемонстровано на рисунку 1.2:

INTERNAL_FREQ_FINE register

FREQ_ FINE7	FREQ_ FINE6	FREQ_ FINE5	FREQ_ FINE4	FREQ_ FINE3	FREQ_ FINE2	FREQ_ FINE1	FREQ_ FINE0

INTERNAL_FREQ_FINE register description

FREQ_FINE[7:0]	Difference in percentage of the effective ODR (and timestamp rate) with respect to the typical. Step: 0.15%. 8-bit format, 2's complement.

The formula below can be used to calculate a better estimation of the actual ODR:
ODR_Actual = (26667 + ((0.0015 * INTERNAL_FREQ_FINE) * 26667))

Рисунок 1.2 - Склад регістру INTERNAL_FREQ_FINE та можливі значення

Для отримання максимального значення потрібно записати у регістр значення 255. Останнім кроком буде визначення значення шкали акселерометра. Для цього необхідно записати значення у біти FS регістру CTRL1_XL. Склад регістру продемонстровано на рисунку 1.3:

CTRL1_XL register description

XL_EN[2:0]	Enables accelerometer: (000: Power-down (default); 101: accelerometer enabled;) All other configurations are not allowed.
FS[1:0]_XL	Selects accelerometer full-scale (see Table 30).
LPF2_XL_EN	Selects accelerometer high-resolution. (0: output from first stage digital filtering selected (default); 1: output from LPF2 second filtering stage selected)

Accelerometer full-scale selection

FS[1:0]_XL	Full scale
00 (default)	±2 *g*
01	±16 *g*
10	±4 *g*
11	±8 *g*

Рисунок 1.3 - Склад регістру CTRL1_XL та можливі значення

Для отримання найбільшої розподільного значення потрібно записати 00.

1.3 Особливості калібрування акселерометра і виставлення OFFSET-значень

Акселерометри є механічними структурами з елементами, що вільно рухаються. Ці рухомі елементи можуть бути дуже чутливими до механічних впливів (ударів, трясок), набагато чутливишими ніж сама електроніка. Зміщення при 0g є важливою метрикою акселерометра, оскільки вона визначає поріг для вимірювання реального прискорення. Додаткові похибки вимірювання виникають при монтуванні системи з акселерометром. Ці похибки можуть бути викликані напруженнями в друкованій платі при монтуванні, застосуванням різних компаундів до компонента. Тому, калібрування рекомендовано проводити після збірки системи, щоб компенсувати їх вплив.

Найпростіший спосіб калібрування акселерометра полягає в усереднені значень вимірів (вибірок) використовуючи так звану схему з єдиною точкою калібрування. В схемі з єдиною точкою калібрування систему з акселерометром орієнтують, так щоб одна вісь, як правило вісь Z, знаходиться в гравітаційному полі 1g, а інші осі X і Y в полі 0g. Рекомендується провести щонайменше 10 вибірок з інтервалом в 0,1 с при частоті вимірювання 100 Гц. Ці значення зберігаються як X_{0g}, Y_{0g}, і Z_{+1g} для 0 g вимірювань по осях X, Y і 1 g вимірювання по осі Z.

Значення виміряні для X_{0g} і Y_{0g} є зміщеннями по осях X і Y. Компенсація здійснюється відніманням цих значень від вихідних значень даних акселерометра, щоб отримати реальне прискорення:

$$X_{ACTUAL}=X_{MEAS}-X_{0g} \quad (1.1)$$
$$Y_{ACTUAL}=Y_{MEAS}-Y_{0g} \quad (1.2)$$

де X_{ACTUAL}, Y_{ACTUAL} – реальні значення прискорення; X_{MEAS}, Y_{MEAS} – виміряні значення прискорення; X_{0g}, Y_{0g} – зміщення (значення при відсутності обертань навколо будь-якої осі).

Оскільки вимірювання по осі Z проведені у гравітаційному полі в +1 g, і схема калібрування припускає ідеальну чутливість S_Z для осі Z. Це значення віднімається від Z_{+1g}, щоб отримати зміщення по осі Z, яке потім віднімати-меться від наступних виміряних значень для отримання реального значення прискорення:

$$Z_{0g}=Z_{+1g}-S_Z \quad (1.3)$$
$$Z_{ACTUAL}=Z_{MEAS}-Z_{0g} \quad (1.4)$$

IIS3DWB може автоматично компенсувати вихід використовуючи значення записані в спеціальні OFFSET-регістри. Регістри OFSX (адрес 0x1E), OFSY (адрес 0x1F), OFSZ (адрес 0x20) призначені для виставлення зміщень по осях X, Y і Z, відповідно. Вміст кожного регістра додається до виміряного значення прискорення по відповідній осі, а результат розміщується в регістрах

даних DATA. Регістр має масштабний коефіцієнт 15,6 mg/LSB і є незалежним від вибраного діапазону вимірювання прискорення. Молодшому значущому біту відповідає прискорення 15,6 mg (коефіцієнт розрахунку 15,6 mg/LSB), причому враховується знак, для значення 0x7F отримаємо приблизно +2g, для 0x80 відповідно –2 g. Оскільки значення розміщене у регістрах додається, то потрібно розмітити в них від'ємне значення, щоб відкинути додатне значення зміщення і навпаки для від'ємного зміщення. Наприклад, припустимо що IIS3DWB налаштовано з максимальною роздільною здатністю 13 біт і чутливістю 256 LSB/g (±2 g). Система орієнтована, таким чином, що вісь Z знаходиться в гравітаційному полі, а виміряні вихідні значення прискорення по осях X, Y і Z є +10 LSB, –13 LSB і +9 LSB, відповідно. Використовуючи попередні рівняння, X0g є +10 LSB, Y0g є –13 LSB, і Z0g є +9 LSB. Кожний молодший значущий біт (LSB) ви-ходу при максимальній роздільній здатності є 3,9 mg або одна четверта значення молодшого біта регістра зміщення. Оскільки значення в регістрі додається, то значення беруться з протилежним знаком і заокруглюються до найближчого значення молодшого значущого біту регістра зміщення.

$$XOFFSET=-Round(10/4)=-3\ LSB \quad (1.5)$$
$$YOFFSET=-Round(-13/4)=3\ LSB \quad (1.6)$$
$$ZOFFSET=-Round(9/4)=-2\ LSB \quad (1.7)$$

Ці значення програмуються у регістри OFSX, OFSY і OFXZ, відповідно, як 0xFD, 0x03 і 0xFE. Як зі всіма регістрами в акселерометрі IIS3DWB, OFFSET-регістри не зберігають записані в них значення при відключенні живлення. Вимкнення і повторне увімкнення живлення до акселерометра повертає OFFEST-регістри до значень за замовчуванням 0x00. Оскільки метод відсутності обертання або одноточковий припускає ідеальну чутливість по осі Z, то нема ніяких помилок в результаті чутливості при помилці зміщення. Наприклад, якщо реальна чутливість є 250 LSB/g, то зміщення буде 15 LSB, але не 9 LSB. Щоб зменшити цю похибку, можуть бути використані додаткові точки вимірювання з віссю Z в полі 0g і 0g вимірювання може бути використано в рівнянні ZACTUAL.

Отже, розроблене апаратне забезпечення базується на використанні МК STM32L476, цифрових трьохосьових акселерометрах IIS3DWB, флеш пам'яті, що забезпечує виконання усіх функцій та характеризується низькою ціною.

1.4 Огляд засобів розробки вбудованого ПЗ

Програмне забезпечення контролера розроблюєтеся на низькому рівні абстракції від апаратного забезпечення. Це обумовлює необхідність застосування специфічних рішень для розробки, таких як ОСРЧ.

Відмінність ОСРЧ від звичних користувачу операційних систем полягає у здатності забезпечити необхідний рівень сервісу в певний проміжок часу.

Ідеальна ОСРЧ має передбачувану поведінку при всіх сценаріях навантаження, включаючи одночасні переривання і виконання потоків.

Порівняльні характеристики ОС загального призначення та ОСРЧ наведені у таблиці 1.

Таблиця 1 – Порівняльні характеристики ОС

	ОС реального часу	ОС загального призначення
Основна задача	Встигнути зреагувати на події, що відбуваються на устаткуванні	Оптимально розподілити ресурси комп'ютера між користувачами та задачами
На що орієнтована	Обробка зовнішніх подій	Обробка дій користувача
Позиціонування	Інструмент для створення конкретного апаратно-програмного комплексу реального часу	Сприймається користувачем як набір застосунків, готових до використання

Операційна система, яка забезпечує необхідний час виконання завдання навіть в найгірших випадках, називається операційною системою жорсткого реального часу. Система, яка може забезпечити необхідний час виконання завдання в середньому, називається операційною системою м'якого реального часу.

Системи жорсткого реального часу не дозволяють затримок реакції системи, так як це може призвести до втрати актуальності, значний фінансових втрат а також аварій і катастроф. Ситуація, в якій обробка подій відбувається за час, більший передбаченого, в системі жорсткого реального часу вважається помилкою. При виникненні такої ситуації операційна система перериває операцію і блокує її, щоб не постраждала надійність і готовність іншої частини системи.

1.5 Обґрунтування вибору засобів розробки програмного забезпечення контролера

Розробка програмного забезпечення для мікроконтролерних пристроїв має деякі відмінності від розробки ПЗ прикладного рівня. Високі вимоги до продуктивності і необхідність гарантувати строго певний час реакції на події визначають необхідність роботи на низькому рівні абстракції від апаратного забезпечення.

Важливою частиною розробки представляється розгортання середовища розробки та вибір програмних засобів для реалізації пристрою. STMicroelectronics, який виробляє контролери STM32, пропонує розробнику широкий вибір інструментальних засобів розробки вбудованого ПЗ.

Середовище розробки Atollic TrueSTUDIO було створене на базі популярної платформи з відкритим вихідним кодом - Eclipse. Програма включає в себе: редактор коду (з підтримкою мов C / C ++ і асемблера), менеджер проектів, C / C ++ компілятор і систему збирання вихідного коду (з можливістю паралельної компіляції), відладчик (з підтримкою функції многоядерного налагодження), а також різні додаткові інструменти для: аналізу коду (відповідно до стандарту кодування MISRA-C), управління завданнями, відстеження змін / оновлень, пристрій автоматично перевіряти створеного коду.

Підтримує всі необхідні інструменти для розробки і налагодження. Є можливість аналізу пам'яті мікроконтролера, налагодження в процесі виконання, відладчик дозволяє переглядати стан RTOS об'єктів. Є можливість покрокової налагодження під час виконання.

До 2018 року середовище розробки була платною, але після покупки компанією STMicroelectronics, стала доступною як для комерційних проектів, так і для освіти.

Графічний інтерфейс користувача програмного продукту Atollic Truestudio приведений на рисунку 1.5.

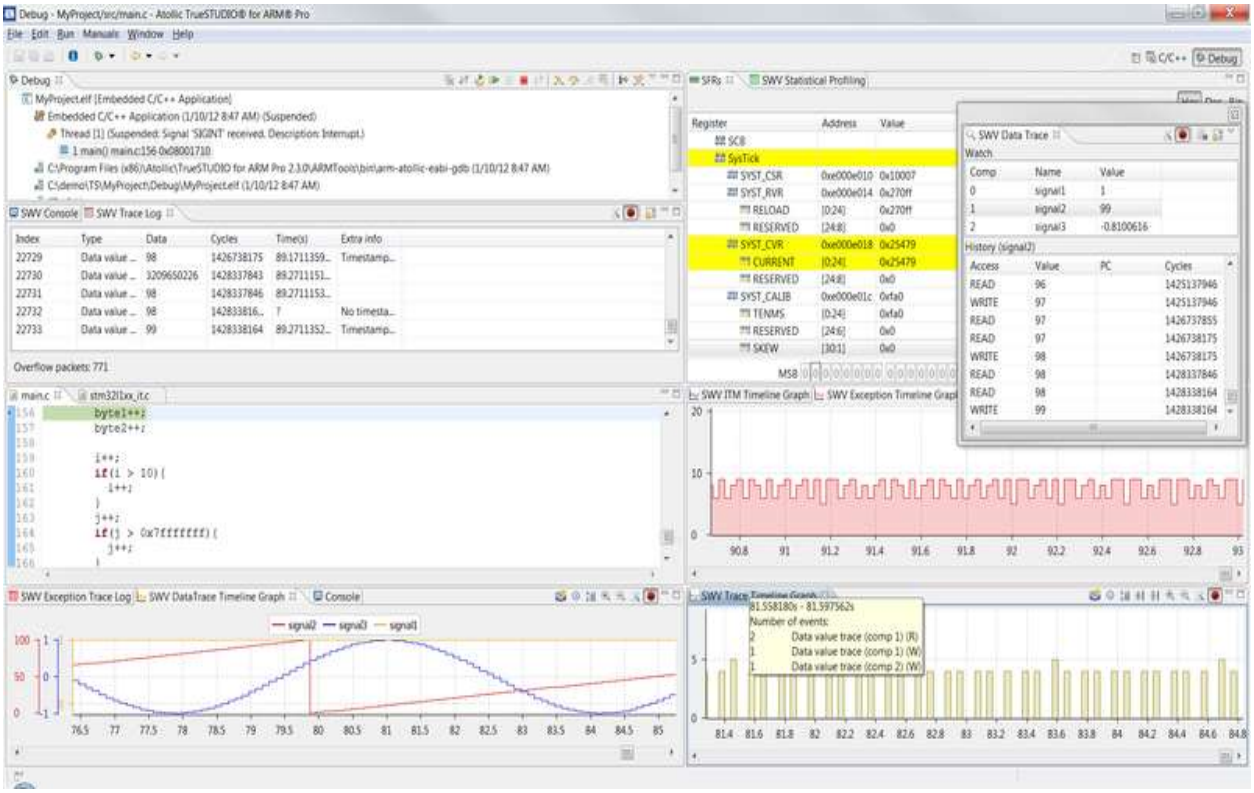

Рисунок 1.5 – Графічний інтерфейс Atollic Truestudio

Середовище програмування розроблене компанією Keil, яка була заснована в Мюнхені в 1982 році. У жовтні 2005 року Keil увійшла до складу американської корпорації ARM. На сьогоднішній день вона представляє широкий спектр різних засобів для розробки програм, що включають C-

компілятори, макроассемблера, відладчики, симулятори, лінкери, IDE-додатки і оціночні плати для різних сімейств мікроконтролерів.

Середовище Keil uVision – комерційне програмне забезпечення, ціна для юридичних осіб складає приблизно дві тисячі доларів. Безкоштовна версія має обмеження на розмір кінцевого файлу прошивки – 32кб.

Графічний інтерфейс користувача програмного продукту Keil uVision приведений на рисунку 1.6:

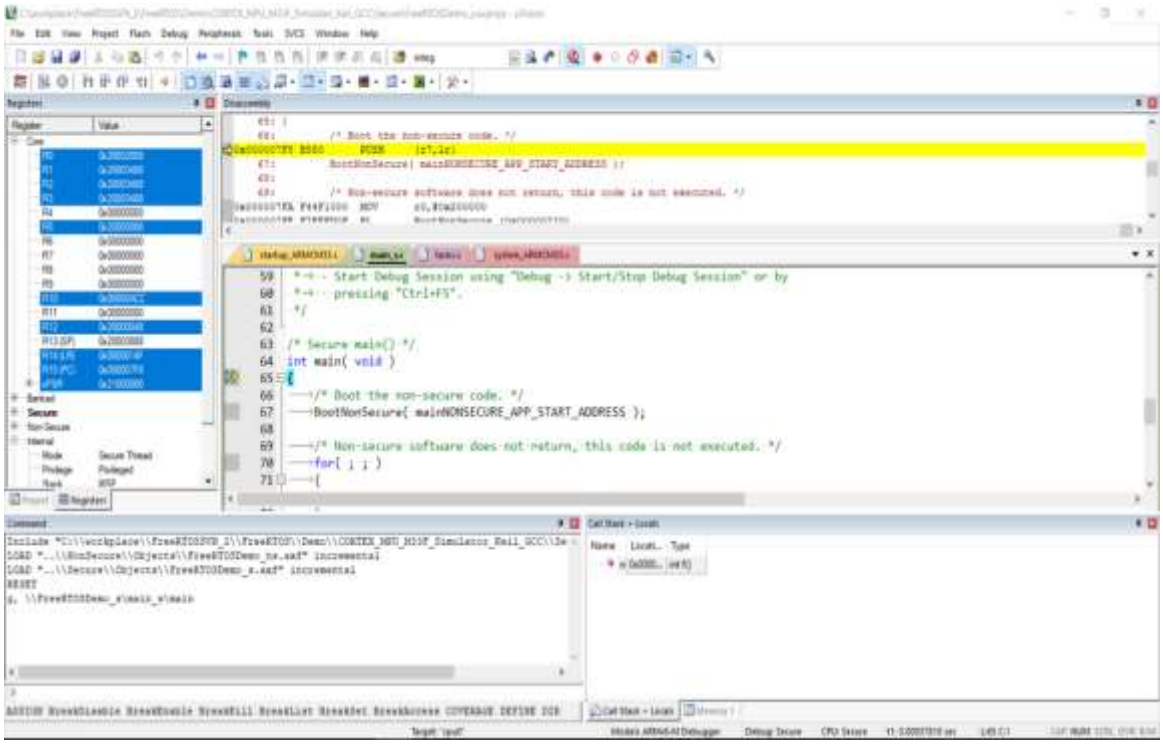

Рисунок 1.6 – Графічний інтерфейс Keil uVision

IAR Embedded Workbench — інтегроване середовище розробки (IDE) випущене фірмою IAR Systems. Містить у собі зручний інтерфейс, оптимізовану CLIB/DLIB бібліотеку, підтримує різноманітні RTOS (Micrium uC/OS-II, OSEC ORTI), а також JTAG- адаптери різних фірм (OLIMEX, Phyton, ASHLING). IAR Embedded Workbench підтримує широкий спектр 8-, 16-, 32-розрядних мікроконтролерів — ARM, Actel, Infineon, NEC, Cypress, Atmel, Micronas, Analog Devices, ZiLOG, Microchip, Luminary Micro, Maxim, OKI, NXP, Samsung, STMicroelectronics, Texas Instruments, Renesas, Freescale Semiconductor, SiLabs і т.д. Кожній платформі відповідає своє середовище, наприклад платформі ARM відповідає IAR Embedded Workbench for ARM, платформі 8051 - IAR Embedded Workbench for 8051. В комплект IAR Embedded Workbench входять: C/C++ компілятор, транслятор мови асемблера, компонувальник, підпрограми для роботи з бібліотеками, редактор, менеджер проектів, C-SPY відладчик. Комерційний продукт, вартість якого близька до вартості Keil uVision.

Графічний інтерфейс користувача програмного продукту IAR Embedded Workbench приведений на рисунку 1.7.

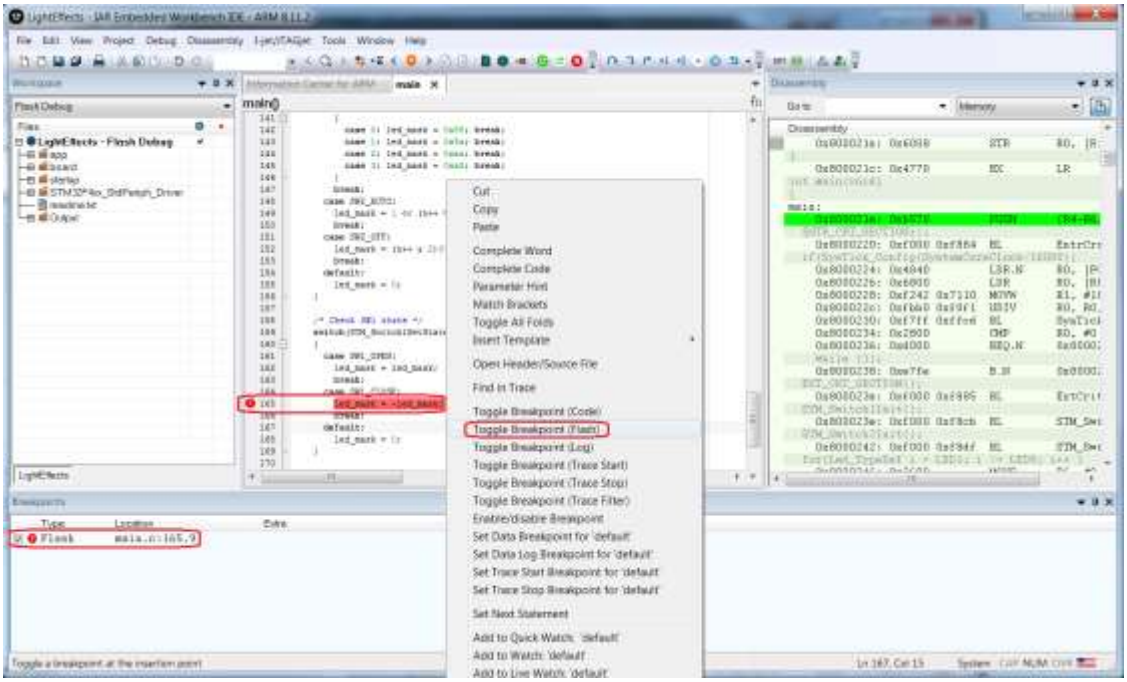

Рисунок 1.7 - Графічний інтерфейс користувача IAR Embedded Workbench

Виходячи з опису доступних програмних продуктів була складена таблиця, яка містить основні преваги та недоліки представлених інтегрованих середовищ розробки**.** Кожен з атрибутів може бути оцінений за п’ятибальною шкалою. Рішення про використання буде прийматися виходячи з підсумковоі оцінки.

Таблиця 1.2 – Порівняння середовищ розробки

	IAR	KEIL	Atollic
Внутрисхемна відладка	5	5	4
Менеджер пам’яті	5	5	5
Робота з RTOS об’єктами	4	5	5
Компілятор	5	5	4
Ціна	0	0	5
Загальна оцінка	19	20	23

Для некомерційних та освітніх проектів вибір програмного продукту Atollic є оправданим. Повнофункціональне середовище розробки надається безкоштовно, на відміну від конкурентів

1.6. Архітектура вбудованого ПЗ

Реалізація програмного забезпечення, що працює в реальному часі, можлива при використанні апаратних засобів контролера. Такими засобами є таймери. Використовуючи таймер і механізм переривань можна реалізовувати механізми збору і обробки даних через потрібні проміжки часу.

Для більш ефективної роботи в складі контролера буде працювати операційна система реального часу. При використанні такої системи реалізація системи спрощується, так як завдання розробляються незалежно один від одного. У готової системі завданням встановлюється пріоритет, виходячи з якого планувальник передає управління під час роботи. Операційна система реального часу зберігає контекст завдання. Приклад розподілу часу представлений на рисунку 1.8.

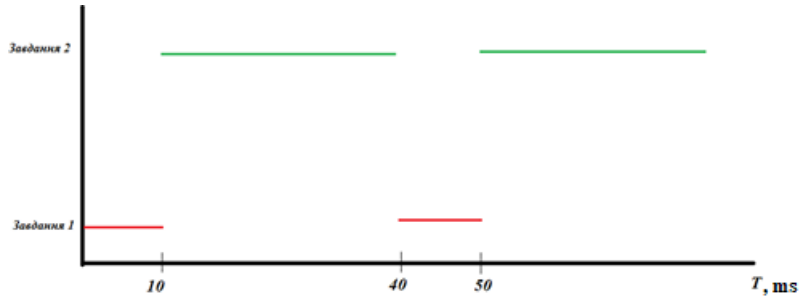

Рисунок 1.8 – Приклад розподілу часу між завданнями

Алгоритм роботи ОСРЧ: Завдання контролера має пріоритет вище, ніж завдання інтерфейсу. Такий розподіл виконано виходячи з цільового призначення. Завдання контролера виконується 10мс і після виконання переходить в блоковане стан на 30мс. Після переходу завдання 1 в блокований стан управління отримує завдання 2, яка виконується 40мс. Після 30мс планувальник передає управління завданню 1, зберігаючи при цьому контекст завдання 2. Після закінчення завдання 1 управління знову отримує завдання 2, починаючи виконання з моменту зупинки. Завдання 2 завершивши своє виконання через 10мс переходить в блоковане стан на заданий проміжок часу.

1.7 Обґрунтування вибору архітектури програмного забезпечення контролера

Програмне забезпечення контролера розроблюєтеся на низькому рівні абстракції від апаратного забезпечення. Це обумовлює необхідність застосування специфічних рішень для розробки, таких як ОСРЧ.

Відмінність ОСРЧ від звичних користувачу операційних систем полягає у здатності забезпечити необхідний рівень сервісу в певний проміжок часу[3]. Ідеальна ОСРЧ має передбачувану поведінку при всіх сценаріях навантаження, включаючи одночасні переривання і виконання потоків.

Порівняльні характеристики ОС загального призначення та ОСРЧ наведені у таблиці 1.3.

Таблиця 1.3 – Порівняльні характеристики ОС

	ОС реального часу	ОС загального призначення
Основна задача	Встигнути зреагувати на події, що відбуваються на устаткуванні	Оптимально розподілити ресурси комп'ютера між користувачами та задачами
На що орієнтована	Обробка зовнішніх подій	Обробка дій користувача
Позиціонування	Інструмент для створення конкретного апаратно-програмного комплексу реального часу	Сприймається користувачем як набір застосунків, готових до використання

Операційна система, яка забезпечує необхідний час виконання завдання навіть в найгірших випадках, називається операційною системою жорсткого реального часу. Система, яка може забезпечити необхідний час виконання завдання в середньому, називається операційною системою м'якого реального часу.

Системи жорсткого реального часу не дозволяють затримок реакції системи, так як це може призвести до втрати актуальності, значний фінансових втрат а також аварій і катастроф. Ситуація, в якій обробка подій відбувається за час, більший передбаченого, в системі жорсткого реального часу вважається помилкою. При виникненні такої ситуації операційна система перериває операцію і блокує її, щоб не постраждала надійність і готовність іншої частини системи.

1.8 Аналіз отриманих значень віброприскорення

1.8.1 Виявлення викидів

Викид - це спостереження або набір спостережень, які значно відхиляються від того, що вважається нормальним. Визначення нормальної поведінки пов'язано з тим, як розподіляються дані, що зазвичай визначається на основі історичних даних. Детектор викидів може порівнювати нові спостереження з загальним розподілом моделі або з розподілом підмножини найостанніших спостережень. Якщо різниця між новим спостереженням і моделлю значна, то детектор буде вважати це спостереження викидом. До проблеми виявлення викидів можна підійти кількома способами. Один з підходів - моделювати тільки нормальну поведінку. Інший підхід - моделювати як нормальне, так і ненормальну поведінку.

1.8.2 Виявлення змін

Виявлення змін - це проблема виявлення змін розподілу ймовірностей випадкової величини або тимчасового ряду. Як правило, нові спостереження слідують в послідовному порядку в потоках даних часових рядів. Точка зміни - це перехід між різними розподілами даних. Така зміна даних може бути пов'язана з відхиленням від норми або нормальними змінами які раніше не спостерігалися в системі, яка виробляє дані. Основна характеристика цього типу виявлення полягає в тому, що зміна є постійною і не обмежується одним викидом або набором викидів. Приклади аномалій наведено на рисунку 1.9:

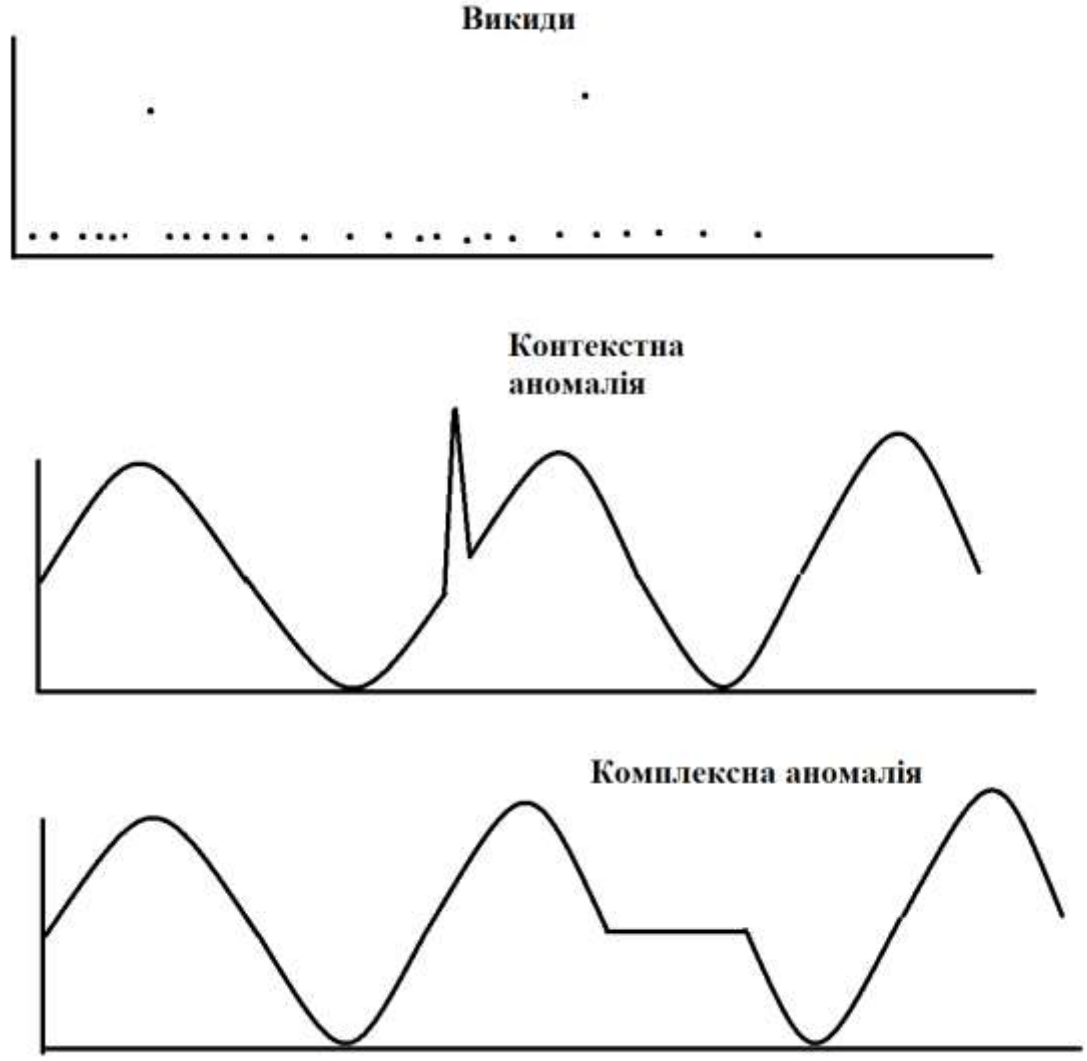

Рисунок 1.9 - Приклади різних типів аномалій

1.8.3 Точкова аномалія

Точкова аномалія - це індивідуальне вимір, яке не відповідає розподілу інших даних. Це найпростіший і, мабуть, найпоширеніший тип аномалії.

1.8.4 Контекстна аномалія

Аномалія цього типу також відома як умовна аномалія і відноситься до вимірювання, яке є аномальним в певному контексті, але не в інших. Контекст визначається загальною або локальною структурою даних. В цьому випадку окремий вимір може вважатися нормальним при певних умовах, але аномальним в інших умовах.

1.8.5 Колективна аномалія

Колективна аномалія виникає, коли група вимірювань аномальна по відношенню до нормального розподілу даних. Індивідуальне вимір саме по собі не є колективною аномалією. Однак, коли окремий вимір розглядається з групою вимірювань, набір вимірювань може бути колективної аномалією.

Зазвичай механічна система, така як підшипник, схильна до контекстних або колективним аномалій, враховуючи характер і тимчасової аспект даних, доступних від датчиків, які контролюють ці системи. Система, що працює довгий час, зазнає змін через природній знос системи. Такий знос відноситься до зміни, яке вважається нормальною поведінкою. Таким чином, в ідеалі метод виявлення аномалій повинен бути здатний враховувати такі зміни і адаптуватися до них, оскільки вони не відповідають аномалій в системі. При розгляді виявлення аномалій в потоці даних виникають дві основні проблеми. Одна з проблем пов'язана з тим, як дані збираються і обробляються. В автономному сценарії дані збираються і обробляються партіями. Навпаки, онлайн-обробка даних вимагає послідовної роботи в реальному часі. Друга проблема пов'язана з обмеженими ресурсами, доступними для виявлення аномалій. У разі потокової передачі даних постійно з'являються нові дані. Таким чином, дані необхідно якось обробляти і агрегувати, що ускладнює завдання аналізу і обробки

1.9 Вимоги до системи вібромоніторінгу обладнання на підприємстві

Згідно з проведеним аналізом літератури та сформованими цілями та завданнями, можно сформувати формалізовані функціональні та нефункціональні вимоги.

1.9.1 Функціональні вимоги

1.9.1.1 Програмне забезпечення має виконувати наступні функції:
- 1.9.1.1.1 збір даних з датчиків вібрації;
- 1.9.1.1.2 Відображення даних та формування звітів;
- 1.9.1.1.3 мережева взаємодія;

1.9.1.2 Програмне забезпечення контролера повинно працювати у режимі реального часу, відповідно до обмежень апаратної платформи.

1.9.1.3 Програмне забезпечення повинно формувати наступні звіти:

1.9.1.3.2 Звіти та повідомлення про стан обладнання;
1.9.1.3.3 Діаграма відображення показників датчиків .

1.9.2 Нефункціональні вимоги

1.9.2.1 Програмне забезпечення повинно підтримувати українську мову
1.9.2.2 ПЗ повинно забезпечувати стабільну доступність Web-інтерфейсу при умові на явності з'єднання.
1.9.2.3 ПЗ повинно коректно обробляти помилки, попереджати і видавати повідомлення про несподіваних ситуаціях.
1.9.2.4 Браузеры: Google Chrome, Yandex, Mozilla FireFox, Internet Explorer.
1.9.2.5 Швидкість Internet не меньшу 200 кбит/с.
1.9.2.6 Рівень користувача – середній (оператор).
1.9.2.7 Вимоги до апаратної частини:
1.9.2.6.1 Вимоги до ПК оператора:
1.9.2.6.1.1 процессор 2 ядра по 2 ГГц чи вище;
1.9.2.6.1.2 оперативна пам'ять – 1ГБ чи більше;
1.9.2.6.1.3 вільної пам'яті – 100 МБ чи більше;
1.9.2.6.1.4 пристрої маніпулювання з користувачем - клавіатура і миша;
1.9.2.6.1.5 вимоги до ОС –Windows 10.
1.9.2.6.2 Вимоги до апаратного забезпечення контролера:
1.9.2.6.2.1 ARM M4 – ARM M7 сумісний контролер;
1.9.2.6.2.2 стабильна робота у режимі реального часу;
1.9.2.6.2.3 підтримка RTOS;
1.9.2.6.2.4 CC2650 сумісний контролер мережевого обміну.

1.10 Висновоки з першого розділу

У результаті виконання аналізу літератури та публікацій були розглянуті основні методи проведення вібродіагностики обладнання а також особливості проектування вбудованого програмного забезпечення. За результатами аналізу методів оцінки та обробки показників було прийнято рішення реалізувати хмарне рішення та бездротову архітектуру датчиків.

Загальні характеристики, яким повинна відповідати система:

1) Бездротовий інтерфейс передачі даних;
2) Робота у режимі реального часу;
3) Обчислення та упаковка значень на самому датчику;
4) Генерування повідомлень про помилки та аварії обладнання.

2 ПЛАНУВАННЯ ЕКСПЕРИМЕНТУ ДЛЯ ВИЗНАЧЕННЯ ВІДПОВІДНОСТІ СИСТЕМИ ВІБРАЦІЙНОІ ДІАГНОСТИКИ ОБЛАДНАННЯ

2.1 Мета експерименту

.

Мета експерименту – визначити відповідність розробленого програмно-апаратного рішення визначеним у першому розділі вимогам. Потрібно зазначити важливі для розроблюваної системи параметри:

1) Автономність та енергоспоживання апаратної частини
2) Якість функціонування авто калібрування акселерометра
3) Якість функціонування бездротової мережі BLE

2.2 Побудова моделі системи вібродіагностики обладнання

Апаратне забезпечення системи побудоване на МК STM32L476 і трьохосьовому цифровому акселерометрі IIS3DWB . Акселерометр IIS3DWB встановлюється на об’єкті моніторингу (наприклад на фрезерній головці фрезерувального верстата з ЧПК) і підключається по шині SPI до МК. Мікроконтролер збирає дані з давача та їх опрацьовує. Акселерометр IIS3DWB використано як давач для вимірювання вібрацій. IIS3DWB – це мініатюрний трьохосьовий цифровий акселерометр фірми STMicroelectronics з малим енергоспоживанням, високою роздільною здатністю (16 біт) і діапазоном вимірювання прискорення до ±16g. Причому діапазон вимірювань можна вибрати з ряду: ±2g, ±4g, ±8g і ±16g. Результат вимірювань можна прочитати побайтно через цифровий інтерфейс SPI (3-х або 4-х провідний) або I2C у вигляді 16-бітних даних.

IIS3DWB відносять до класу ємнісних акселерометрів зі смугою пропускання від 0,05 ... 6000 Гц. Цей прилад є ідеальний для вимірювання динамічних прискорень, низькочастотних вібрацій, статичних прискорень гравітації, руху і кутів нахилу. Смуга пропускання характеризує здатність давача помічати зміни прискорення, що відбуваються з високою частотою (наприклад, вібрація з частотою 1000 Гц). На цю характеристику впливає частота дискретизації вбудованого АЦП акселерометра, яка повинна бути як мінімум в два рази більше смуги пропускання. Максимальна частота дискретизації для IIS3DWB складає 25600 Гц.

В IIS3DWB є можливість вибору роздільної здатності. Фіксована роздільна здатність в 16 біт і режим максимальної роздільної здатності, коли роздільна здатність збільшується зі зростанням діапазону прискорення g. Максимум – до 16 біт при вимірюванні прискорення ±16 g з постійною чутливістю – 4 mg/LSB у всіх діапазонах вимірювання g. Давач має функції виявлення одиночного та подвійного поштовхів, контролю активності/не активності. Функція виявлення вільного падіння; інтерфейси SPI (3-х і 4-х провідний) і I2C; можливість гнучкого задання режимів переривання з

вибором будь-якого (з 2-х можливих) виводів переривань; діапазон вимірювання, також як і смуга пропускання, вибирається подачею відповідної команди. Давач має широкий робочий температурний діапазон від –40 °С до +85 °С і високу ударостійкість до 10000g.

Така вбудована система постійно відслідковує вібрацію, наприклад працюючого верстату, в режимі реального часу і аналізує параметри вібрації. Система видає попереджувальні повідомлення або зупиняє верстат у випадку виникнення неприпустимих вібрацій – запобігаючи таким чином можливі поломки і аварії. Система також має надавати достатню інформацію користувачу, щоб він зміг розпізнати можливі проблеми і прийняти профілактичні заходи на основі аналізу спектру вібрації характерного для конкретного верстата.

Основні переваги такої системи на рівні управління виробництвом полягають в наступному: запобігання несправностей шпинделя і верстата в цілому, захист ріжучого інструменту, а також оброблювальної деталі. Система запобігає нанесенню збитків особливо важливим вузлам верстата з використанням функції швидкого реагування на удар. Така система моніторингу може бути вбудованою в систему ЧПК. Система в змозі своєчасно попередити про ймовірну несправність, що дає можливість користувачу здійснити своєчасні профілактичні заходи.

Моніторинг сумарної вібрації має здійснюватися як в часовому, так і в частотному діапазонах. В часовому діапазоні система безперервно відслідковує стан верстата в режимі реального часу, на основі широкосмугового вимірювання віброшвидкостей, віброприскорень і вібропереміщень.

В частотному діапазоні може бути встановлений ряд переналаштовувальних границь попереджуючих і аварійних сигналів для різних частотних діапазонів. Ці діапазони покривають весь спектр вібрації і дають змогу провести аналіз рівня вібрацій в контрольованій області на предмет перевищення допустимих границь по всьому частотному спектрі.

Система дає змогу виконати аналіз вібрації у часовому діапазоні в трьох окремих діапазонах частот. Перший – це низькочастотний діапазон з регульованою шириною смуг для визначення границі вібропереміщень всередині вказаного діапазону. Другий діапазон середніх частот з регульованою шириною смуг для визначення границі віброшвидкості всередині вказаного діапазону. Третій високочастотний діапазон з регульованою шириною смуг для визначення границі віброприскорень всередині вказаного діапазону.

Смуги можуть бути вибрані на основі конфігурації верстата, умов роботи і характеристик очікуваних несправностей.

2.3 Система вібраційної діагностики

2.3.1 Узагальнена архітектура системи вібраційної діагностики

Узагальнена варіант загальної архітектури представлені на рисунку 2.1:

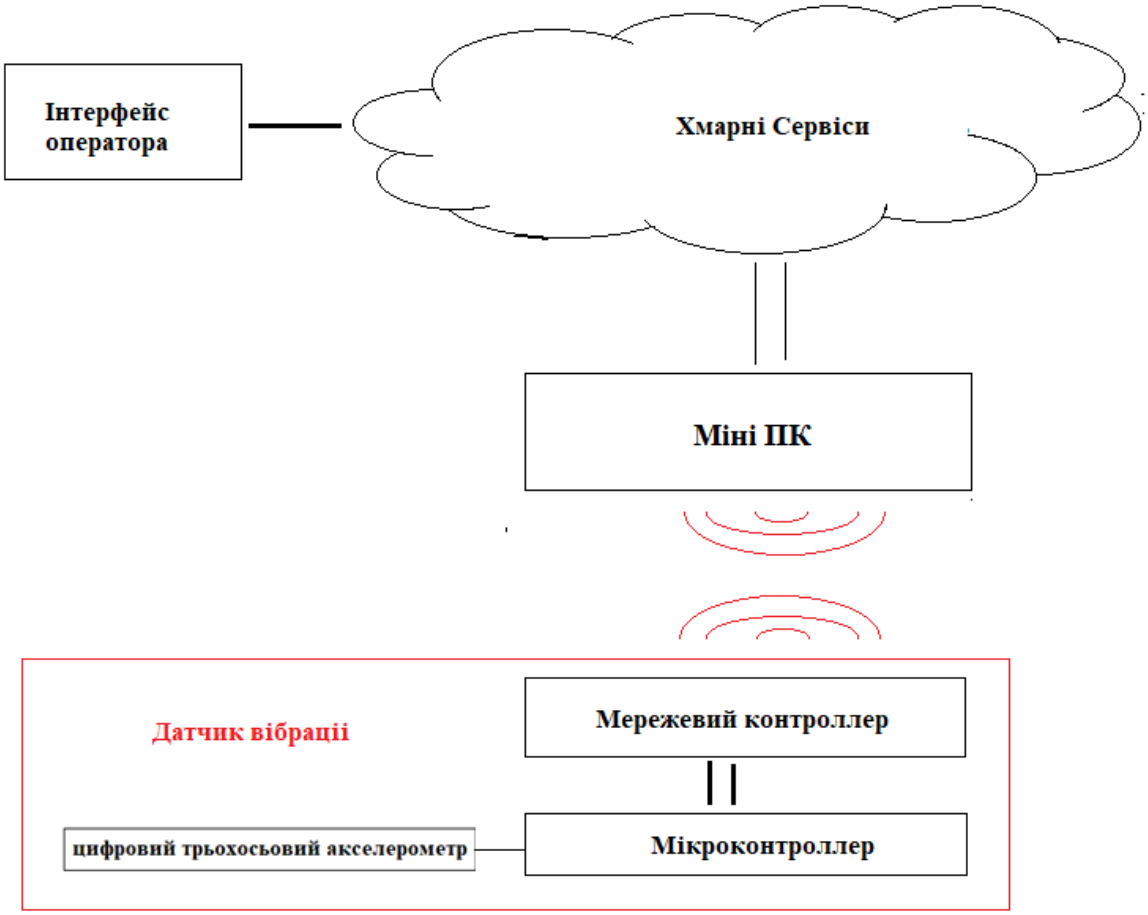

Рисунок 2.1 – Структура апаратної системи

2.3.2 Розроблення датчику вібраційної діагностики

Розроблена структура системи наведена на рисунку 2.2 та включає такі основні складові: мікроконтролер STM32L476 який призначений для збору і опрацювання даних з давача вібрації (акселерометра); трьохосьовий цифровий акселерометр IIS3DWB фірми ST, який використано в якості давача віброприскорень. FRAM пам'ять для буферу; інтерфейси бездротової комунікації для віддаленого обміну інформацією з HUB пристроем. Структура наведена на рисунку 2.2:

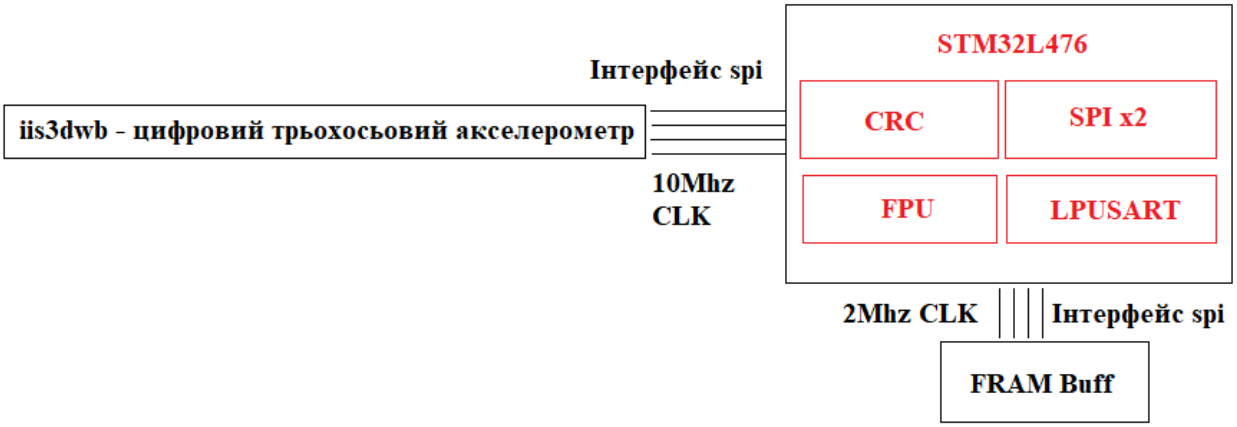

Рисунок 2.2 – Структура апаратної системи

Розроблений алгоритм функціонування системи включає такі кроки (рис. 2):
Крок 1. Ініціалізація акселерометра IIS3DWB на шині SPI .
Крок 2. Зчитування значення з регістра WHO_AM_I і перевірка номера акселерометра.
Крок 3. Налаштування акселерометра IIS3DWB.
Крок 4. Встановлення значень дискретизації та режиму переривань
Крок 5. Встановлення діапазону вимірювання та розподільної здатності
Крок 6. Старт вимірювання
Крок 7. Запис значень у буфер по перериванню та очікування заповнення буферу .
Крок 8. Після заповнення буферу використання FFT для переведення ряду у частотне представлення
Крок 9. Передача за допомогою BLE
Крок 10. Після передачі початок виконування з кроку 6

Блок схему алгоритму наведено на рисунку 2.3:

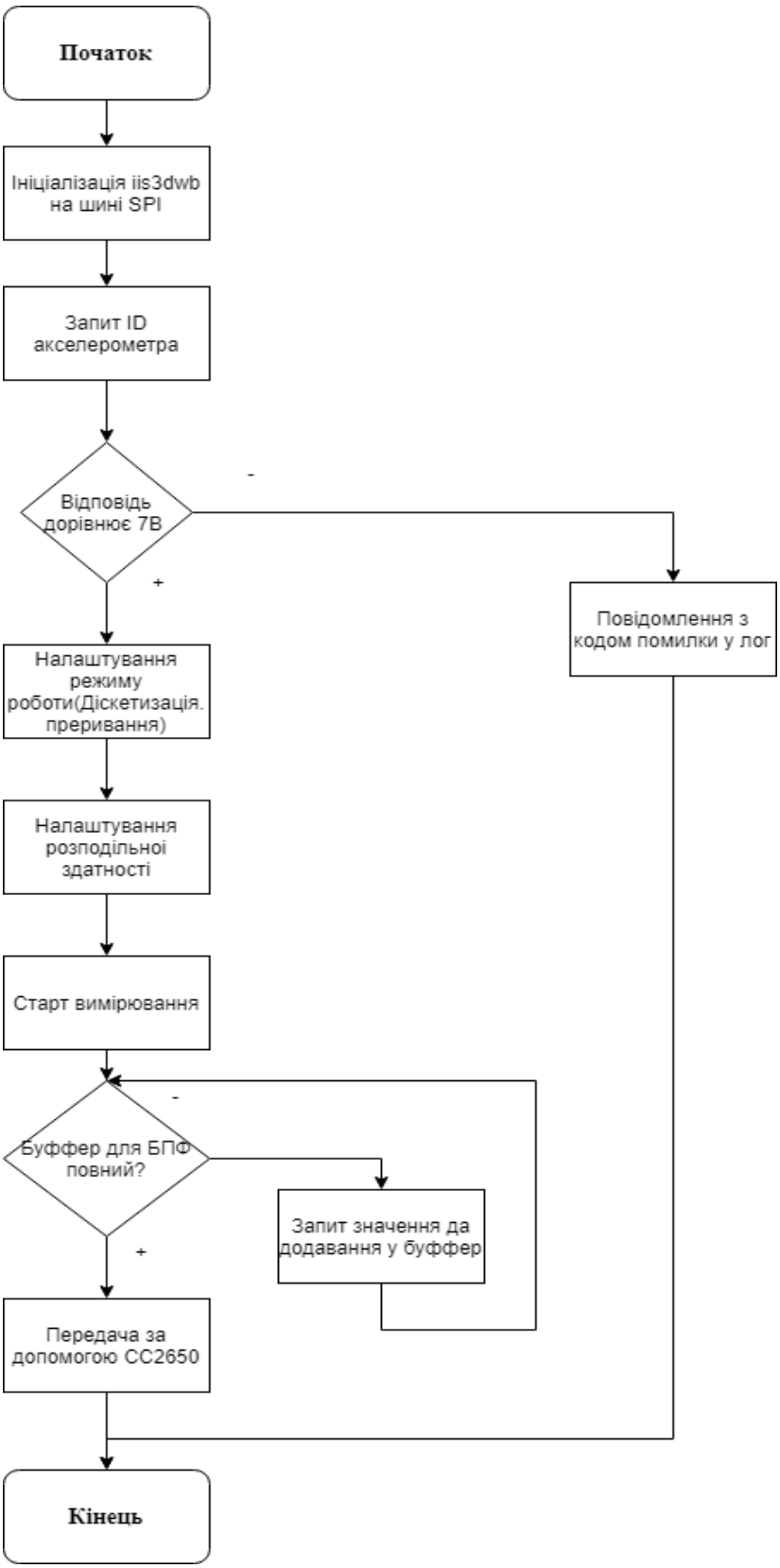

Рисунок 2.3 – Блок схема алгоритму роботи апаратної частини

2.3.3 Реалізація транспортного рівня(HUB рівень)

Так як датчики використовують інтерфейс Bluetooth для обміну даними потрібний пристрій який може отримувати пакети даних від датчику за допомогою Bluetooth та відправляти ці данні за допомогою мережі Internet на віддалений сервер. Цей пристрій також може використовуватися для проміжних обчислень. За основу цього пристрою був взятий мінікомп’ютер з відкритою документацією beaglebone ai. Ви можете побачити мінікомп’ютер beaglebone ai на рисунку 2.4:

Рисунок 2.4 – Мінікомп’ютер beaglebone ai

Вибір даного мінікомп’ютера зумовлений тим що цей проект є відкритим. Це дозволить самому при необхідності дообляти та виробляти цей мінікомп’ютер. Також даний мінікомп’ютер розрахований на обробку сигналів за домогою DSP(Digital Signal Processor) та прискорювачів нейроних мереж.

Робота з датчиками вібраційної діагностики проходить за допомогою Bluetooth модулю який також має данний мінікомп’ютер. Для доступу у мережі Інтернет може бути використаний WiFi та Internet.

Beaglebone ai працює під управлінням Linux а саме Debian. Для реалізації функціональності роботи з вібраційним датчиком була реалізована програма

на мові програмування C++ . Програма реалізує роботу з BLE, має функції буферізації та передає отримані показники на віддалений сервер. Якщо зв’язок з віддаленим сервером відсутній то програма накопичує отриманні вібраційні показники у пам’яті та при відновленні зв’язку передає їх разом з міткою часу. Схему взаємодії хаба можливо побачити на рисунку 2.5:

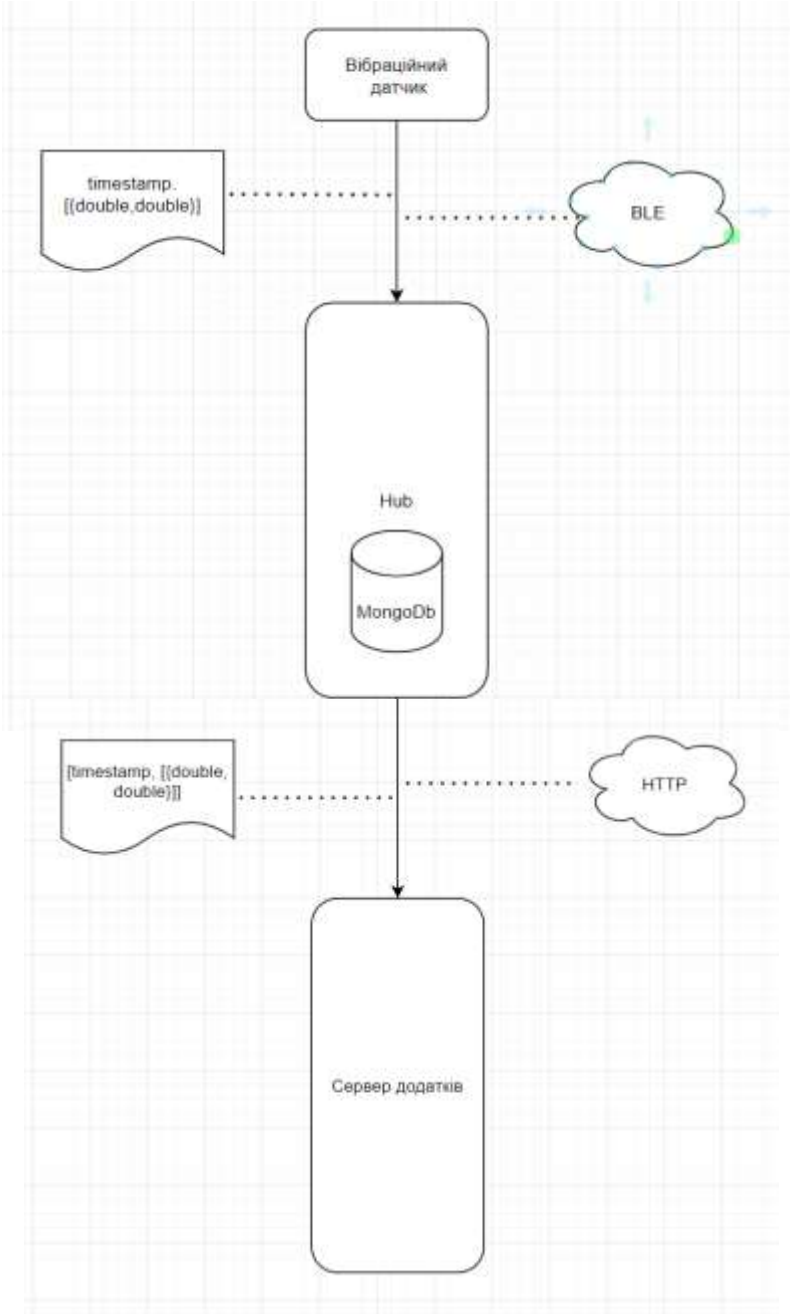

Рисунок 2.5 – Схема взаємодії датчиків з хабом

При передачі між датчиком на хабом використовується пакет наступного вигляду – {timestamp, [{double, double}] } цей пакет містить мітку часу та дані у частотній області які представлені за допомогою масиву елементів з полями double , double в яких зберігається частота на вібраційне прискорення.

Так як датчик вібрації сам переводить показники з часової області у частотну за допомогою швидкого перетворювання Фур’є ці пакети не займають багато місця та з легкістю можуть бути передані за допомогою BLE

на значні відстані з незначними витратами заряду батареї. Якщо будуть потрібні данні у часовій області (наприклад для аналізу) то можливо відновити данні за допомогою оборотного перетворення Фур'є.

Ці пакети даних хаб передає на сервер додатків, якщо зв'язку з сервером немає то хаб починає накопичувати показники у буфер який реалізований за допомогою NoSQL бази даних MongoDb. У подальшому хаб сам зможе аналізувати ці показники за допомогою прискорювачів нейроних мереж.

Для оновлення забезпечення датчиків також використовується хаб. Хаб посилає запроси щоб перевірити теперішню версію забезпечення. Якщо версія відрізняється від тієї версії яку мають датчики підключенні до хабу то хаб починає закачку нової версії. Коли версія закачана перевіряється контрольна сума. Якщо сума співпадає то хаб починає оновлення програмного забезпечення датчиків. Побачити схему оновлення можливо на рисунку 2.6:

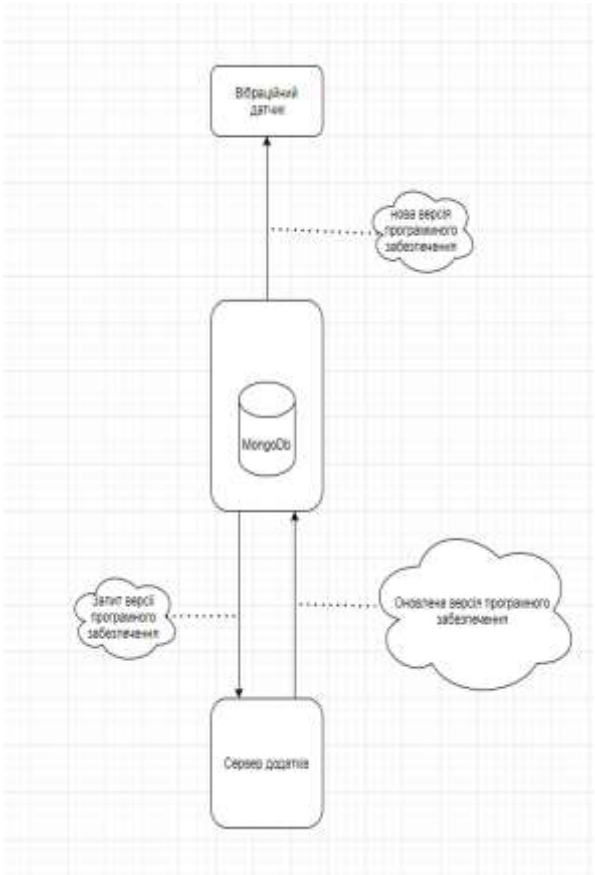

Рисунок 2.6 – схема оновлення ПЗ вібраційного датчика

2.3.4 Засоби розробки алгоритму оцінки стану обладнання

При розробці алгоритму оцінки стану в першу чергу потрібно було досягти мінімальну потрібність алгоритму у навчанні. Тобто щоб оцінка стану проходила точно при мінімальній потрібності у розмічених даних. Для цього алгоритм потрібен мати змогу ідентифікувати патерни та донавчатись на вхідних нерозмічених даних. Подібні техніки називають System Identification.

Приймаючі до уваги вищеописані вимоги та аналіз поточного стану проблеми визначеного у першому розділі за основу був прийнятий алгоритм на базі Dictionary Learning.

Для реалізації алгоритму була використана мова програмування Python так як вона має значну бібліотеку засобів для аналізу сигналів, роботи з нейронними мережами та багату функціональність. Так як при побудуванні системи ми дотримувалися мікро сервісної архітектури то алгоритм оцінки стану обладнання також представлений у вигляді окремого сервісу. Це дозволяє легко та зручно масштабуватись, додаючи нові екземпляри сервісу. Сервіс оцінки не зберігає стану залежного від інших сервісів тому таких екземплярів може бути скільки завгодно екземплярів при тому не потрібно синхронізувати стан цих екземплярів. Кожен екземпляр сервісу знаходиться у окремому контейнері Docker. Це дозволяє швидко розгортувати екземпляри сервісу як на Windows, Linux так і в кластерах з Kubernetes.

При розробці алгоритму використовувався JupyterLab. Це інтерактивне веб-середовище розробки для блокнотів, коду і даних Jupyter. JupyterLab відрізняється гнучкістю: налаштовуйте і упорядковуйте призначений для користувача інтерфейс для підтримки широкого спектра робочих процесів в області науки про дані, наукових обчислень і машинного навчання. JupyterLab є розширюваним і модульним. Є можливість писати плагіни, які додають нові компоненти і інтегруються з існуючими. Побачити інтерфейс JupyterLab можливо на рисунку 2.7.

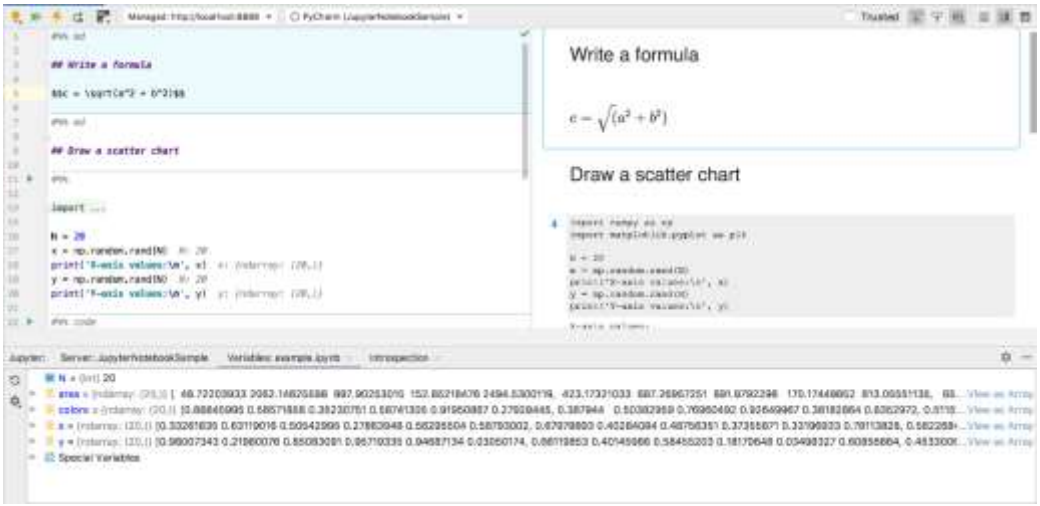

Рисунок 2.7 – Інтерфейс JupyterLab

2.3.5 Датчик вібраційної діагностики

Датчик базується на мікроконтролері STM32L476 з ядром ARM Cortex M4. Цей мікроконтролер включає вбудований високоточний генератор опорної частоти і співпроцесором прискорення математичних операцій з плаваючою крапкою (FPU). Bluetooth реалізований за допомогою Texas Instrument CC2650 в версії MODA. Для отримання вібраційних даних використовується MEMS датчик. Реалізована плата показана на малюнку 2.8.

Рисунок 2.8 – Підсумковий варіант плати

Технічні характеристики:

1) Смуга пропускання: 6kHz;
2) Частота дискретизації: 25.6kHz;
3) Роздільна здатність: 16 біт;
4) Максимальне значення віброприскорення: 2G.

Для обміну даними між MEMS датчиком та мікроконтролером використовується інтерфейс SPI. При розробці ПО для мікроконтролера використовувалася IDE STM32IDE. При розробці плати використовувалося ПЗ KiCad.

2.3.6 Деталізована діаграма системи

Деталізована діаграма системи представлена на рисунку 2.9:

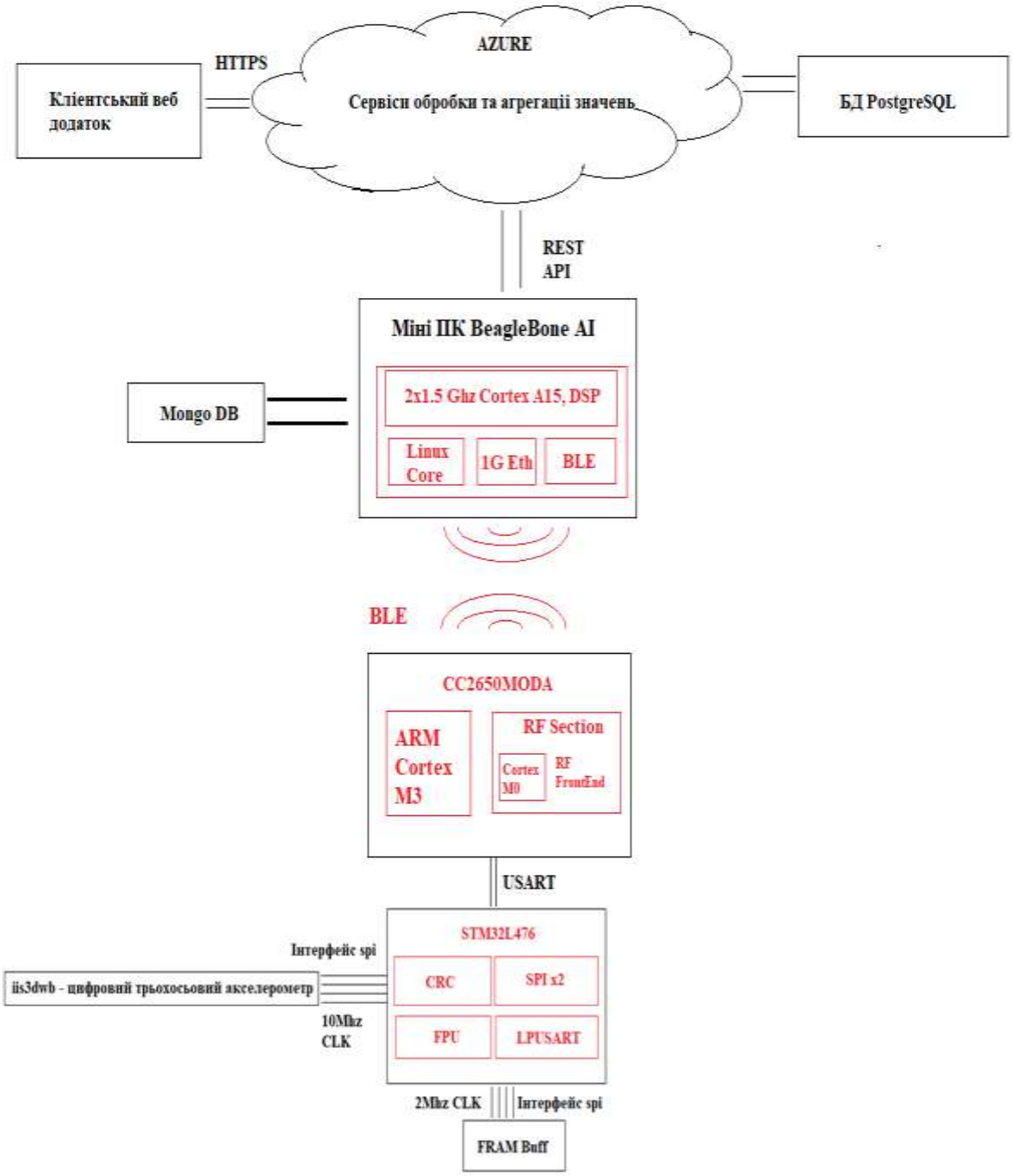

Рисунок 2.9 – Підсумковий варіант архітектури системи

2.4 Вибір інструментальних засобів та обладнання, необхідного для проведення експерименту

Тестування автономності необхідно проводити з метою оцінювання термінів можливого використання апаратної частини без обслуговування. Тестування потрібно проводити у два етапи: тестування енергоспоживання у всіх можливих режимах за допомогою вимірювального обладнання та довгострокове тестування у польових умовах.

Для інструментального тестування недостатньо простого обладнання, наприклад мультиметра, так як пристрій має режим зниженого Енергоспоживання між передачею пакетів. Передача пакету триває кілька десятків мілісекунд, під час якого пристрій споживає кілька десятків міліампер. Після передачі пристрій засинає до наступного циклу, споживаючи одиниці мікроампер. Приклад профілю енергоспоживання пристрою, отриманого під час тестування reference дизайну компанії Texas Instruments наведено на рисунку 2.10:

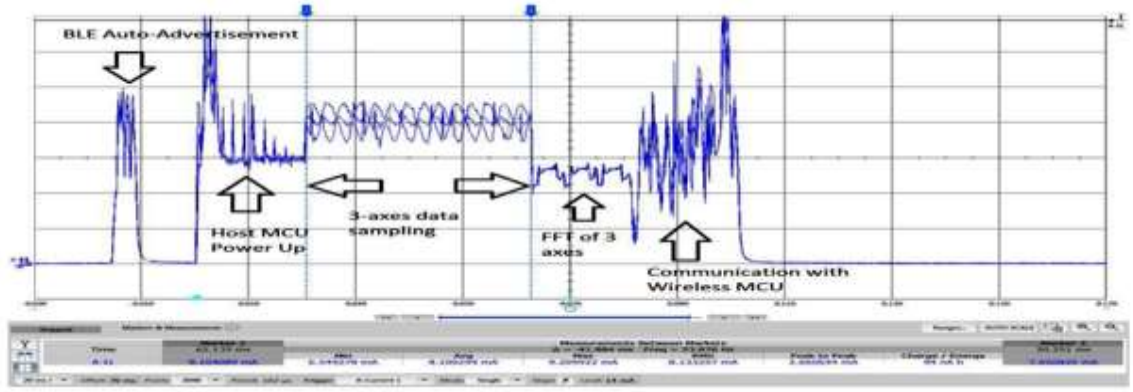

Рисунок 2.10 - Профіль споживання струму датчиком вібрацій на різних етапах роботи

У разі, якщо хост-контролер датчика працює автономно (без бездротового мережевого процесора), основним фактором, що впливає на тривалість автономної роботи, буде період вимірювань. У таблиці 1 наведені оцінки середнього струму споживання і тривалості роботи датчика вібрацій без задіяного BLE-процесора в залежності від періоду вимірювань.

Для таких вимір найкраще підходять спеціалізовані вимірювальні комплекси під назвою «DC Power Analyzer». Прикладом може стати Keysight N6705C. Вимірювальний комплекс представлено на рисунку 2.11:

Рисунок 2.11 - DC Power Analyzer Keysight N6705C.

Такий прилад об'єднує можливості джерела живлення, прецизійного мультиметра і осцилографа. Подібне обладнання дозволить провести вимірювання якісно, проте доступ до такого устаткування утруднений через ціни (починається від 8 тисяч євро, без урахування податків і ціни програмного забезпечення).

Більш поширений варіант - використання осцилографа і додаткового комплекту з шунта і інструментального підсилювача. Необхідність використовувати додаткове обладнання Можна виділити проект з відкритою архітектурою uCurrent. Вигляд uCurrent представлено на рисунку 2.12:

Рисунок 2.12 – Інструментальний підсилювач uCurrent.

Використовувати інструментальний підсилювач необхідно з осцилографом. Використовуючи систему синхронізації осцилографа і

вбудовані математичні функції можна отримати значення спожитого струму за обраний період. Приклад вимірювань представлений на рисунку 2.13:

Рисунок 2.13 – Використання осцилографу для вимірювання споживаного струму.

Такий варіант вимірювання значно доступнішою, але так само вимагає наявності осцилографа. Найбільш доступним варіантом є використання тестової плати з технологією Energy Trace від компанії Texas Instrument. У платах, які підтримують технологію EnergyTrace, програмно-керований перетворювач постійного струму генерує цільове джерело живлення (1,2 В-3,6 В). Часова щільність імпульсів заряду DC DC-перетворювача дорівнює енергоспоживанню цільового мікроконтролера. За допомогою тестової плати за 10 доларів та безкоштовного ПЗ Code Composer Studio можливо отримати значення з частотою 1kHz та роздільною здатністю 1 мікроампер. Приклад інтерфейсу вимірювання зображено на рисунках 2.14 та 2.15:

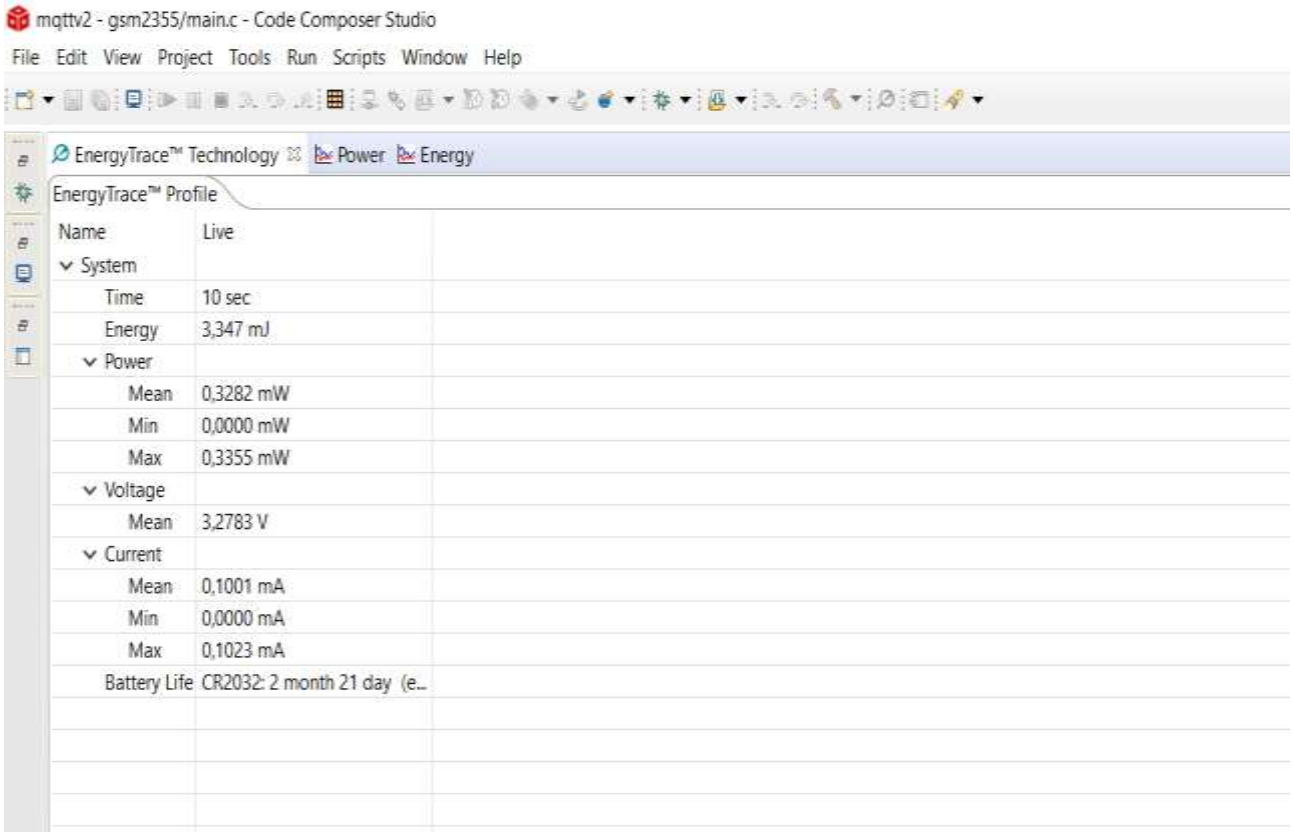

Рисунок 2.14 – Налаштування та результати 10 секундного вимірювання

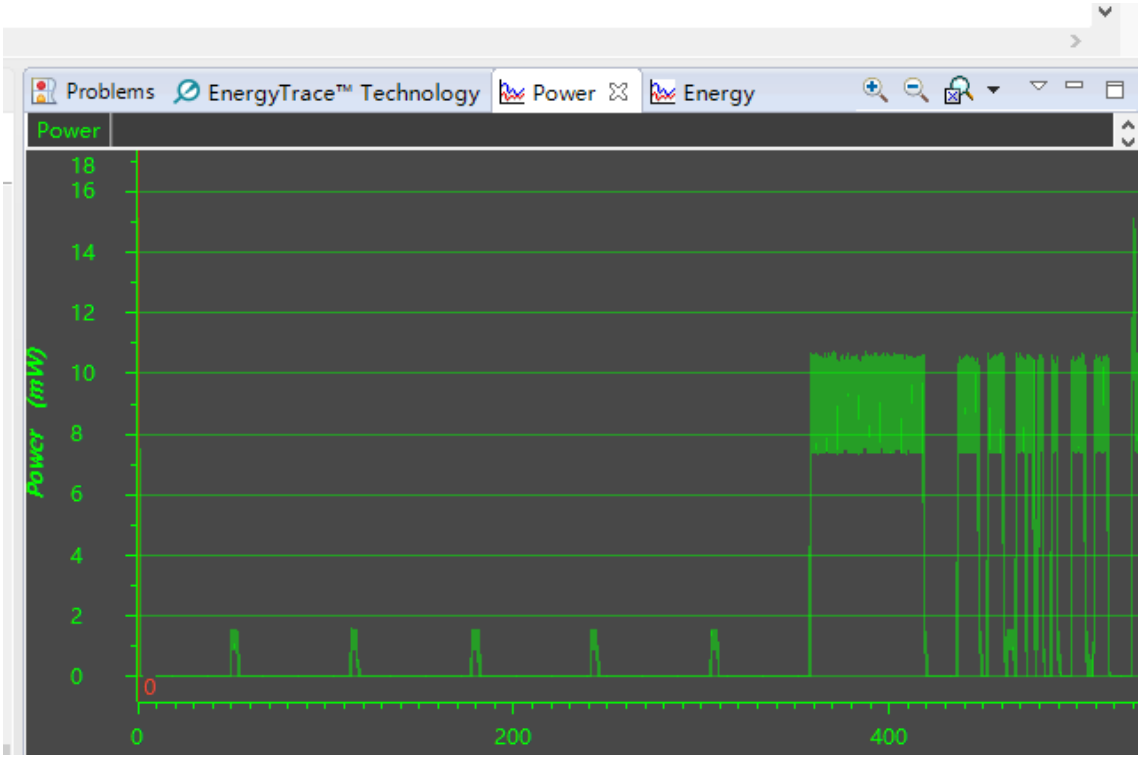

Рисунок 2.15 – Відображення результатів у вигляді графіку споживання

Оцінка функціональності автокалібровки, необхідна для формування виводу про можливості монтажу датчика силами персоналу, не має його якості для цього. Оцінка буде проходити перевірку на стенд відповідності значень еталонного монтажу та монтажу на непідготовлених поверхнях

Вбудовані функції самоперевірки дозволяють перевіряти функціональність пристрою без його переміщення. Коли ввімкнено самотестування акселерометра, до датчика прикладається сила спрацьовування, що імітує певний вхід прискорення. У цьому випадку виходи демонструють зміну своїх рівнів постійного струму, які пов'язані з обраними шкалою через значення чутливості.

Алгоритм роботи функціонала самокалібровки зображено на діаграмі 2.16:

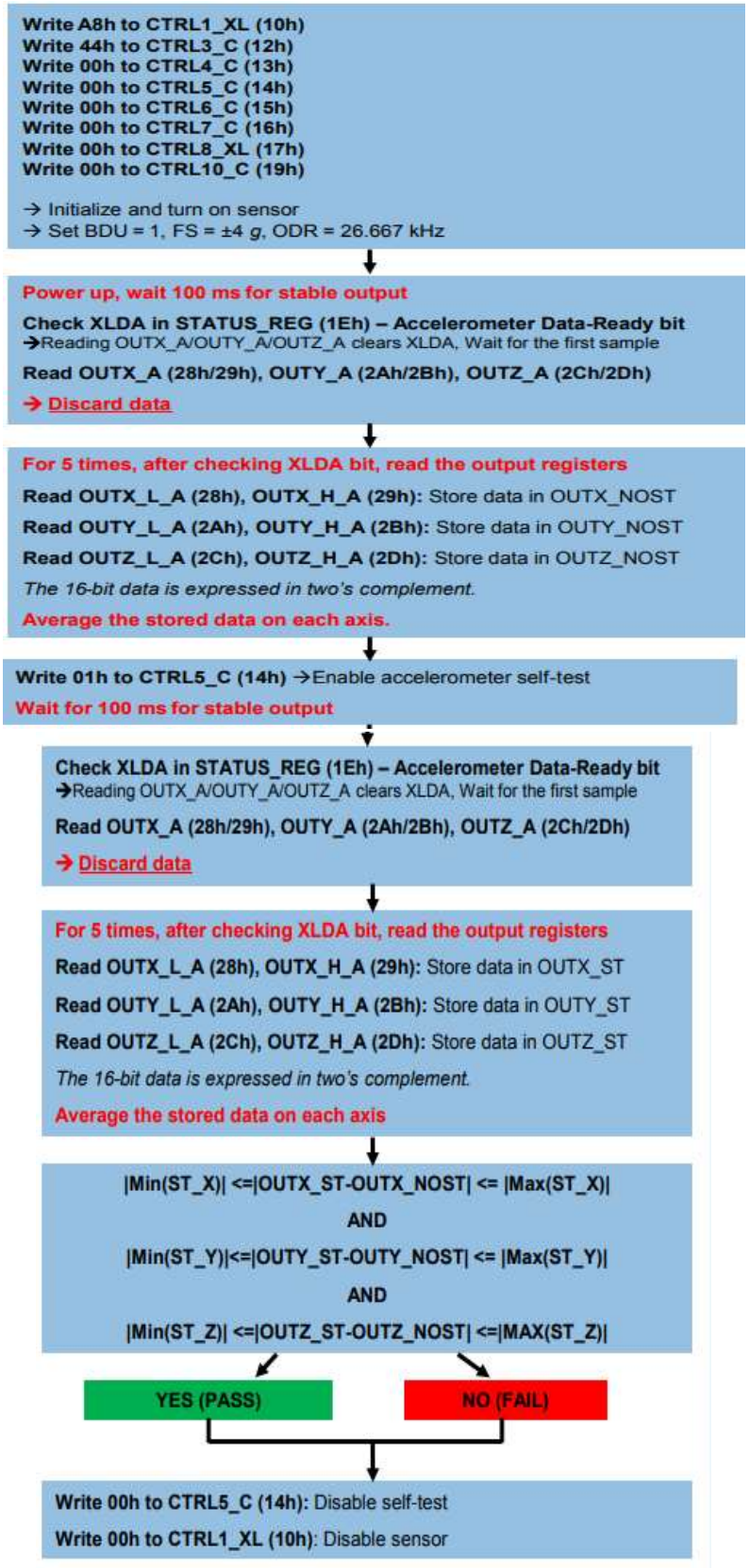

Рисунок 2.16 – Діаграмма роботи режиму автокалібрування сенсору iis3

Якість і стабільність з'єднання бездротової мережі BLE залежить від обраних комплектуючих, якості проектування пристрою і друкованої плати а так само зовнішніх чинників, таких як кількість працюючого обладнання поруч з пристроєм, що тестується. Проводити тестування можна як за

допомогою спеціалізованих інструментальних засобів (аналізатори спектра з відповідними можливостями тестування мереж), так і використовуючи можливості самого протоколу. Мова іде про отримання та аналізу показника рівня сигналу (RSSI).

RSSI (англ. Received Signal Strength Indication) — в телекомунікації, пристрій для вимірювання рівня потужності сигналу. У випадку BLE рівень сигналу дозволяє отримати сама мікросхема трансиверу. Найпростіші схеми розробляються, щоб прийняти вхідний сигнал і сформувати аналогову вихідну напругу (або відповідний цифровий код, одержуваний після подачі цієї напруги на АЦП), пропорційну потужності прийнятого сигналу. Можна використовувати даний показник, щоб оцінити відстань до передавача (для стільникових телефонів — до базової станції).

Як правило, сигнал вимірюється на проміжних частотах перед підсилювачем (наприклад, в стільникових телефонах і інших GSM-пристроях). У пристроях, які працюють без використання проміжних частот, вимір ведеться на основній частоті.

Показник RSSI погано корелює з якістю сигналу, але може використовуватися для приблизної оцінки якості сигналу. Значення вимірюються у **dBm** (опорна потужність 1мВт) по логарифмічній шкалі.

На основі вищеописаних характеристик побудований ряд комерційних технологій навігації в приміщеннях, таргетированій рекламі та системі контролю доступу. Одним з прикладів є технологія маяків iBeacon. Технологія побудована на передачі пакетів, без аутентифікації та механізму встановлення з'єднань, як це було у стандартному Bluetooth. Технологія використовується для:

1) Прив'язка цифрового вмісту до об'єктів фізичного світу
2) Цілісна, добре інтегрована настройка та інтеграція гаджетів
3) Нові концепції для роздрібної торгівлі
4) Інформаційні взаємодії між фізичними особами (англ. Peer-to-peer)

Залежність показань потужності сигналу від відстані, отримана за допомогою проведення експеримент, показана на малюнку 2.17:

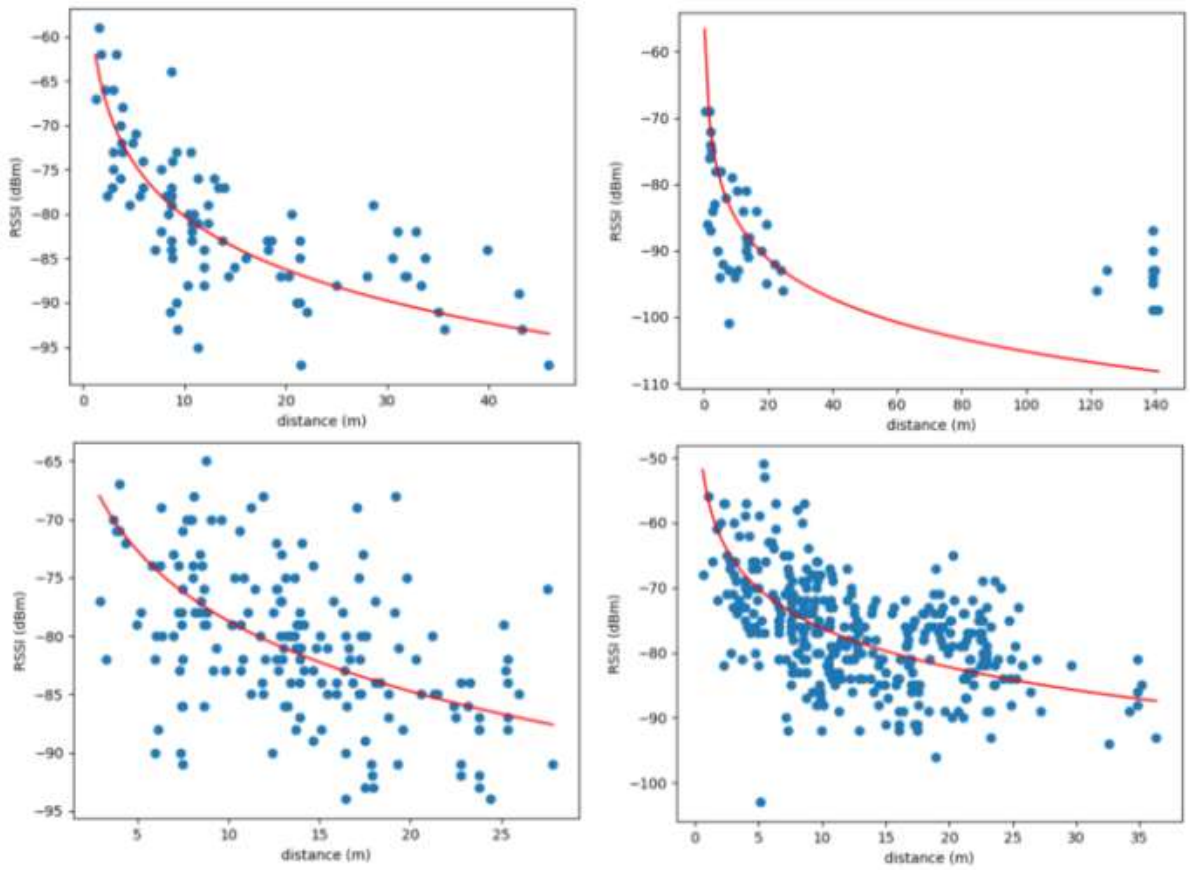

Рисунок 2.17 – Залежність потужності сигналу від відстані

Так є можливість розширеного аналізу не тільки якості передачі сигналів, але і напрямків до джерела сигналів. Змінити виготовлення необхідно для навчання позиціонування датчика.

Методика Angle of Arrival (AoA) замикається у визначенні напрямків, з якого був прийнятий Bluetooth-пакет щодо орієнтації прийнятного узла. AoA зазвичай застосовується при здійсненні методу триангуляції.

Для реалізації технологій використовується набір антен з певними характеристиками, прийом повинен швидко переключатися між окремими антенами, що тимчасово змінюється фазовий здвиг, що приймається сигналом, освічений невеликими різницями в тривалості сигналів для різних антен.

Ці відмінності в тривалості траси будуть залежати від пристроїв, що знаходяться радіочастотних волн щодо антени в решетці. Щоб опрацювати оновлення фазів, пакет Bluetooth повинен мати специфічний вигляд: містити участок безперервного тону (CT), у якому немає фазових сдвигів, виразних модулей.

У AoA-пакетах у польовій навантажувальній упаковці (PDU) передається секція послідовних одиниць, що формують синусоїду на частоті 250 кГц. Це дає змогу встановити час синхронізації, щоб потім вибрати I- (реальна композиційна) та Q- (максимальна складаюча) компоненти сигналів, записати їх у буфер та передати додаток для подальшого аналізу. Структуру пакету представлено на рисунку 2.17:

Структуру пакета для визначення напрямку джерела продемонстровано на рисунку 2.18:

Рисунок 2.18 – Структура пакета для визначення напрямку джерела

При роботі з AOA-пакетами передає радіоядро поміщає тоновий сигнал в секцію PDU-пакета і формує коректну контрольну суму. Приймальна сторона аналізує пакет і починає фіксувати відліки сигналу в потрібний час, синхронізуючи перемикання антен. Відлік зберігаються в пам'яті радіоядра і аналізуються згодом основним ядром.

Вибірка і буферизація відділків відбуваються без участі основного ядра. Завдяки спеціальній попередній обробці додаток на основному ядрі може починати визначення зсуву фаз в сигналі без таких етапів як фільтрація постійної складової і проміжної частоти. Схема взаємодії основного ядра і ядра RF представлена на рисунку 2.19:

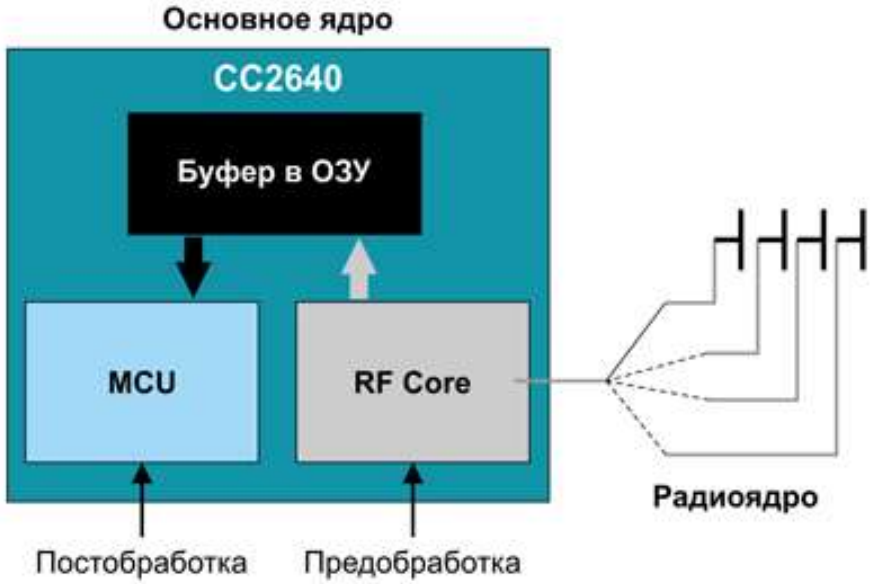

Рисунок 2.19 – Схема взаємодії основного ядра і ядра RF

Для роботи програми потрібні як мінімум два пристрої: провідне, або Master та Slave. Обидва пристрої будуть взаємодіяти в певному діапазоні частот (точніше - частотних каналів) і працювати з певним додатком - списком сінхрослов.

Спочатку відоме пристрій знаходиться в стані прийому на першій із заданих частот і очікує передачі першого в списку сінхрослова. Якщо ведений отримує відповідний пакет, він відповідає пакетом ACK / PONG і змінює частотний канал і сінхрослово за списком, заданим додатком.

Провідний пристрій передає пакет з першим сінхрословом, після чого переходить в режим прийому, чекаючи відповіді від веденого з другим сінхрословом.

Для тестів необхідно апаратне забезпечення, що включає набір антен і схема перемикання трактів. Зовнішній вигляд референс дизайну аналогової плати від TI представлений на малюнку 2.20:

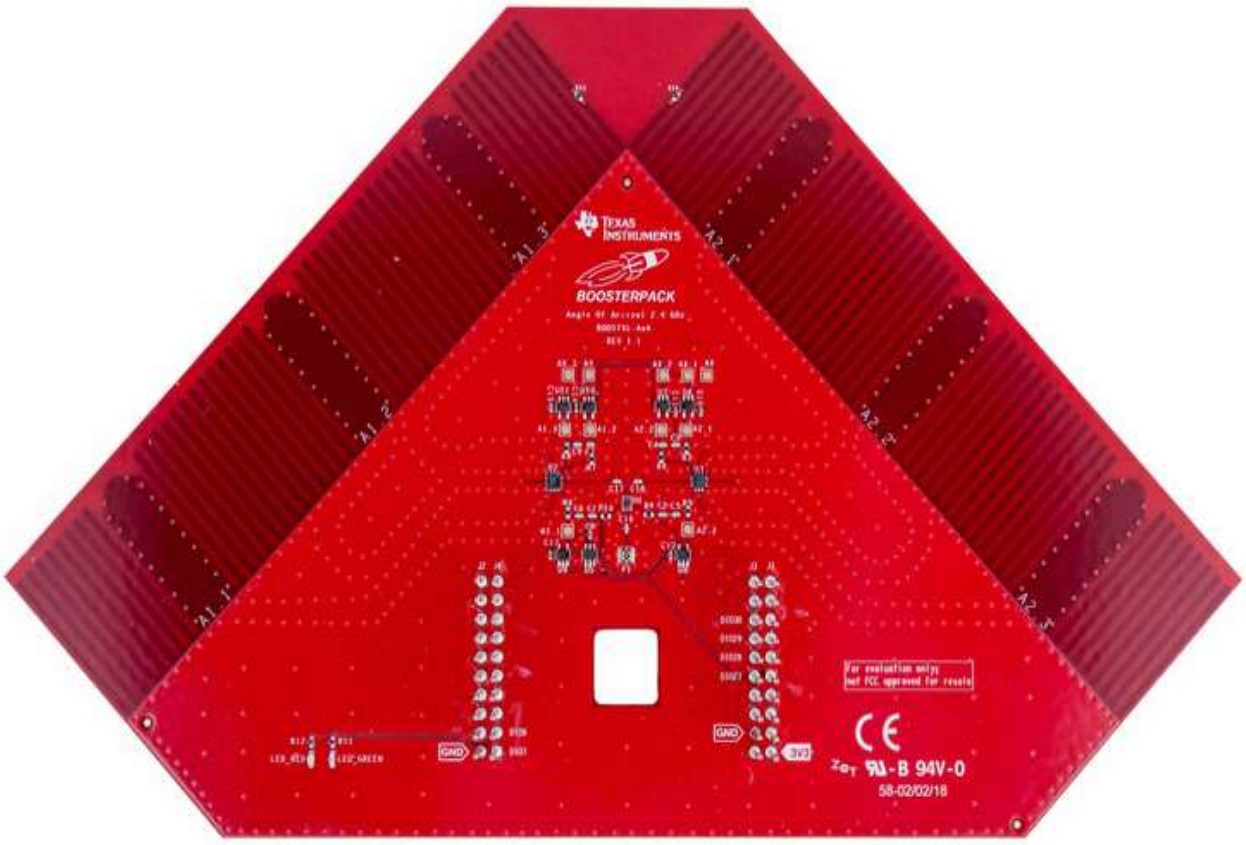

Рисунок 2.20 – Аналоговий frond-end для проведення тестування

2.5 Проектування інструментарію для тестування

Для тестування роботи системи був зроблений стенд що імітує електричний двигун та генератор з’єднаний муфтою.

В корпус стенду закріплений блок живлення та плати керуючі двигунами один з яких імітує електродвигун, а другий генератор. На рисунку 2.21 представлені змонтовані компоненти у корпусі.

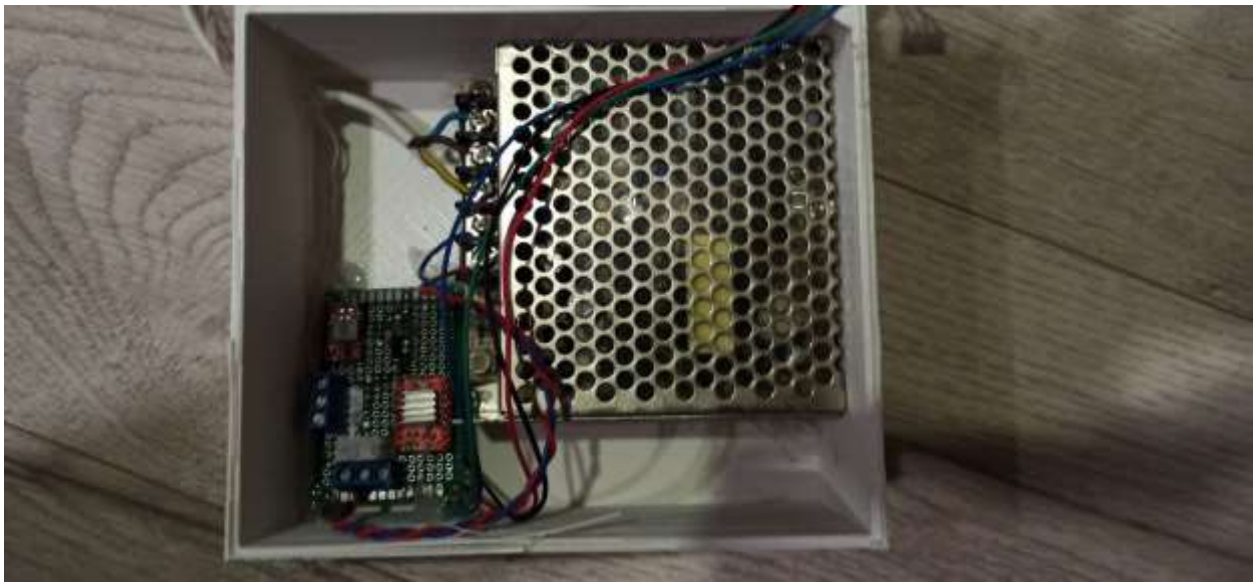

Рисунок 2.21 – Компоненти змонтовані у корпусі

Зверху на корпусі розташовані електродвигуни закріплені у корпусах. Корпус електродвигуна що імітує генератор та створює нагрузку представлений на рисунку 2.22.

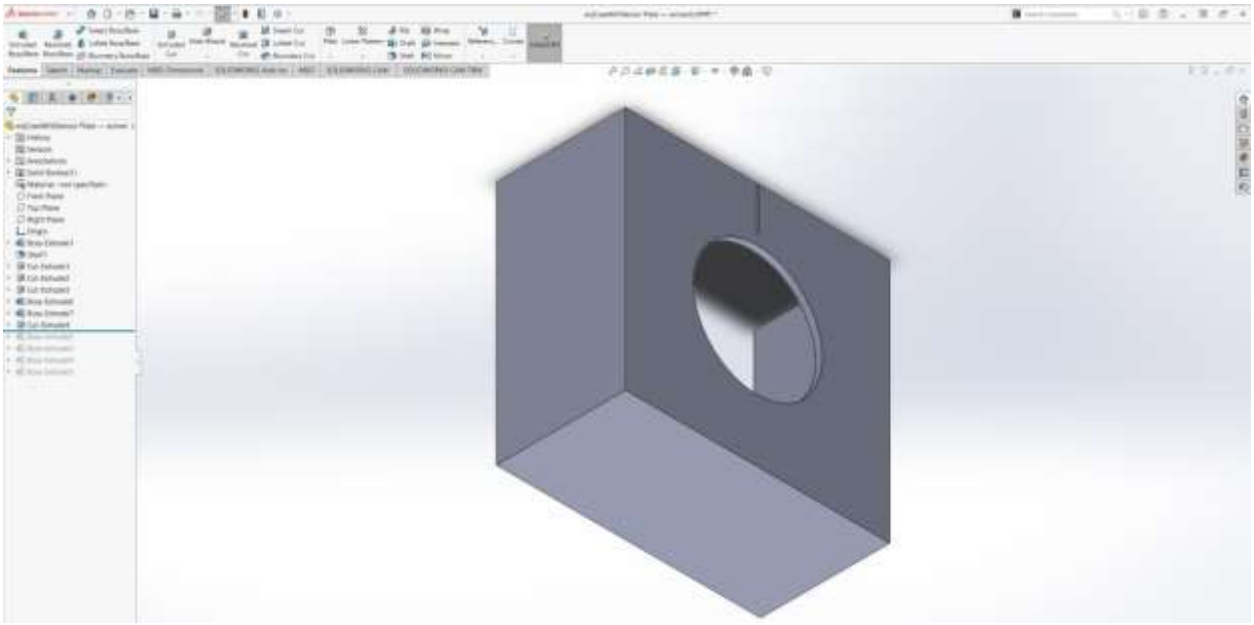

Рисунок 2.22 – Корпус електродвигуна, який створює нагрузку

На корпус другого двигуна який обертається закріплений вібраційний датчик. 3D модель корпусу з місцем кріплення вібраційного датчику представлена на рисунку 2.23.

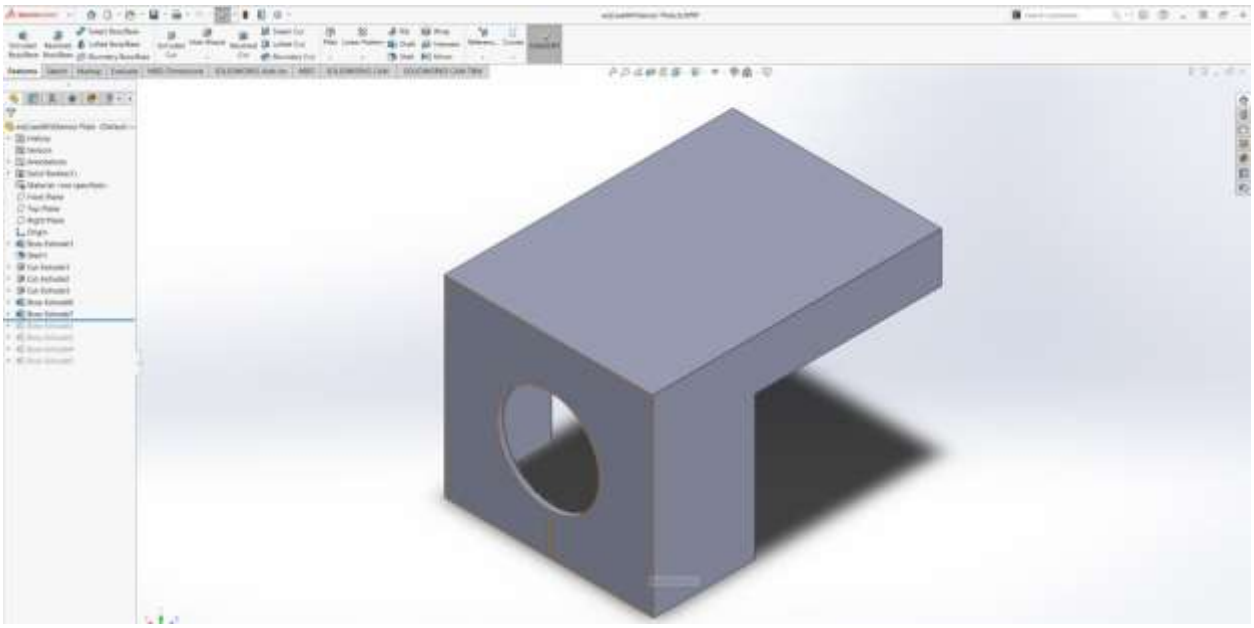

Рисунок 2.23 – Корпус електродвигуна з місцем під кріплення вібраційного датчика

Також для зручного управління стендом створена керуючій пульт з графічним інтерфейсом який дозволяє керувати обертами двигуна. 3D модель корпусу керуючого пульта представлений на рисунку 2.24.

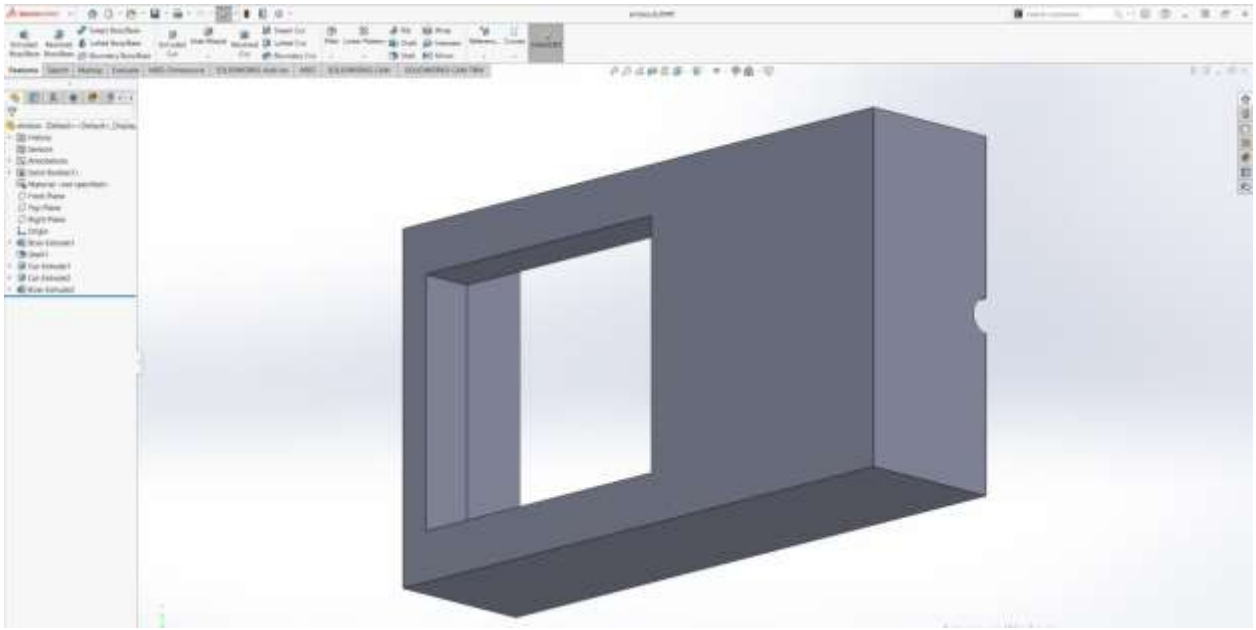

Рисунок 2.24 – 3D модель корпусу керуючого пульту

Зібраний керуючій пульт можливо побачити на рисунку 2.25.

Рисунок 2.25 – Керуючій пульт

Контролер побудований на базі контролеру STM32F437 та має графічний сенсорний екран для відображення інформації та прийняття команд від користувача. Інтерфейс керуючого пульта можливо побачити на рисунку 2.26.

Надрукований на 3D принтері та зібраний стенд представлений на рисунку 2.27.

Рисунок 2.27 – Стенд вид збоку

У складі стенду представлені два крокових двигуна (один з них виконує роль навантаження), контролер крокових двигунів, адаптер змінного струму, тестова плата STM32F4 Discovery, DC-DC перетворювач. Мета розробки стенда - можливість імітування умов роботи на підприємстві. Конструкція стенду покликана імітувати роботу насосного агрегату. Приклад насосного агрегату представлений на рисунку 2.29:

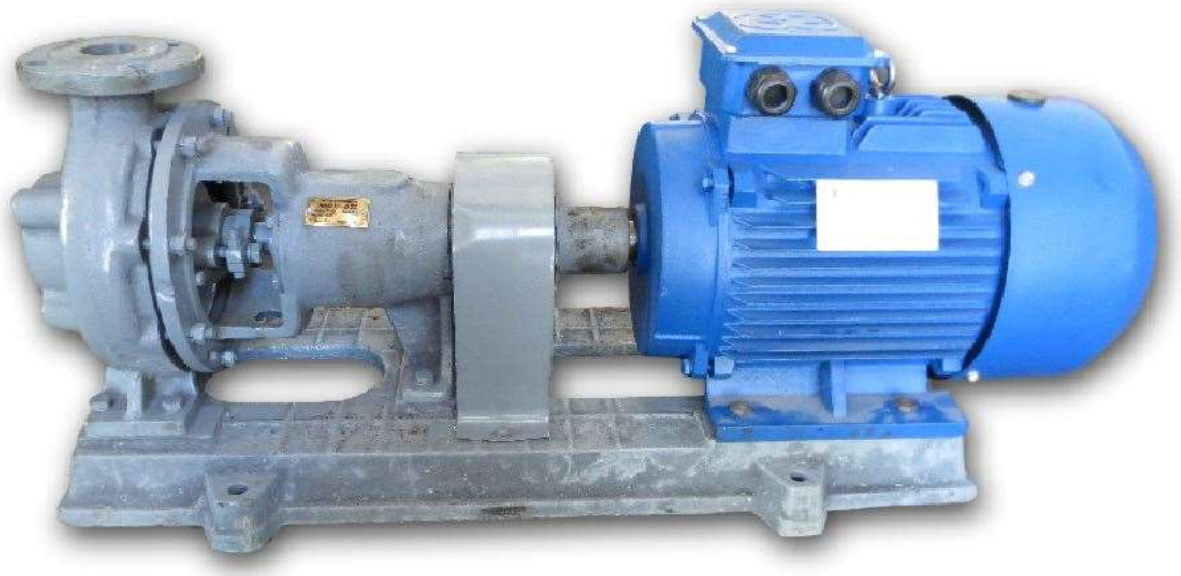

Рисунок 2.29 – Промисловий насосний агрегат

Контроль вібрації в подібних агрегатах проводитися на вузлах навантаження та двигуна. Схематичне уявлення місць кріплення датчиків представлено на рисунку 2.30:

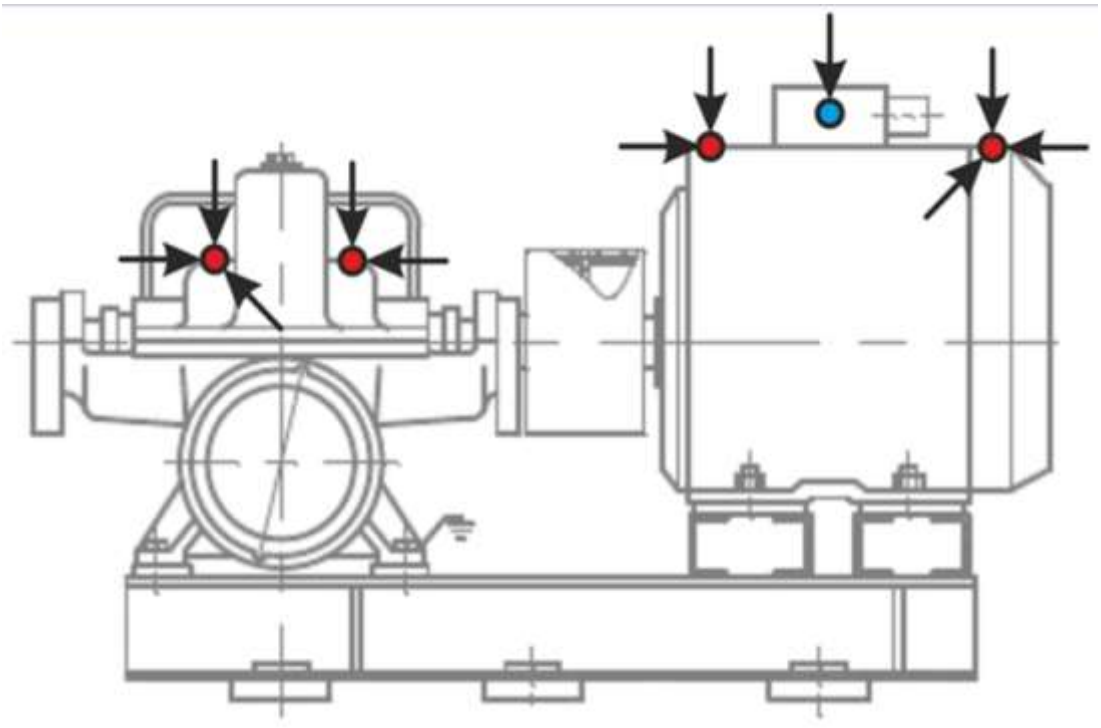

Рисунок 2.30 – точки контролю вібрацій

Одним з переваг обраного складу стенду є можливість регулювати і отримувати показники оборотів не тільки на вбудованому контролері, але і в системі. Це необхідно для зіставлення показників оборотів і отриманих значень датчика вібрацій.

Є кілька варіантів доставки показників швидкості в інформаційну систему. Найбільш доцільним являється перевикористання коду доставки повідомлень датчика вібрації. В якості апаратної платформи використовується CC2650 Launchpad, який зображено на рисунку 2.31:

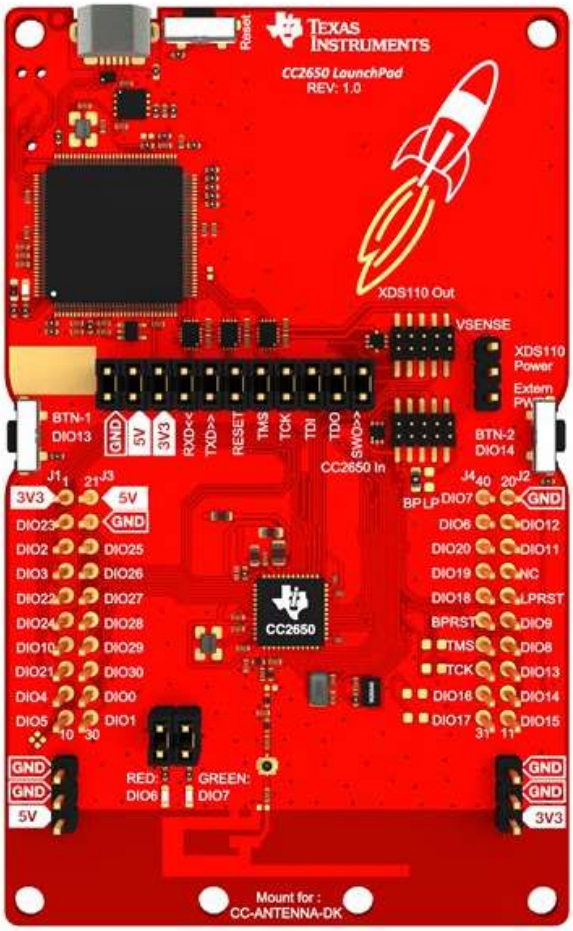

Рисунок 2.31 – CC2650 Launchpad

Розробка програмно-апаратних засобів стенду включає проектування шаблону взаемодействія суті уявлення і суті постачальника даних. Відповідним шаблоном буде Моде́ль–вигляд–контро́лер.

Моде́ль–вигляд–контро́лер - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення[13].

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — дизайн ПЗ, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього

шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами.

Модель (Model) відповідає за зберігання даних та їх структуру. Вигляд (View) відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (у випадку стенда – реакція на натиснення) і передає дані у модель.

Структура отриманого проекту представлена на рисунку 2.33:

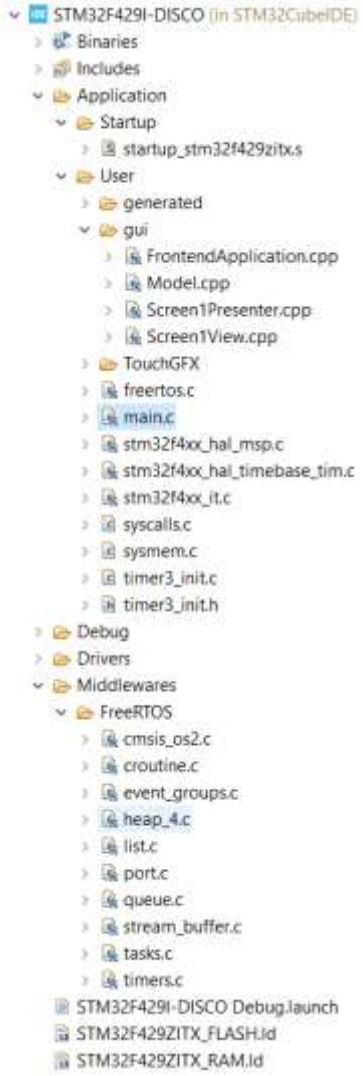

Рисунок 2.33 – Склад проекту стенду

Взаємодія з інформаційною системою відбувається за допомогою протоколу BLE. Архітектура мережевого взаємодії побудована на основі реалізації основного контролера в ролі хоста і BLE системи у вигляді

мережевого контролера. Взаємодія реалізовано з використанням протоколу USART. Реалізація взаємодії представлена на рисунку 2.34:

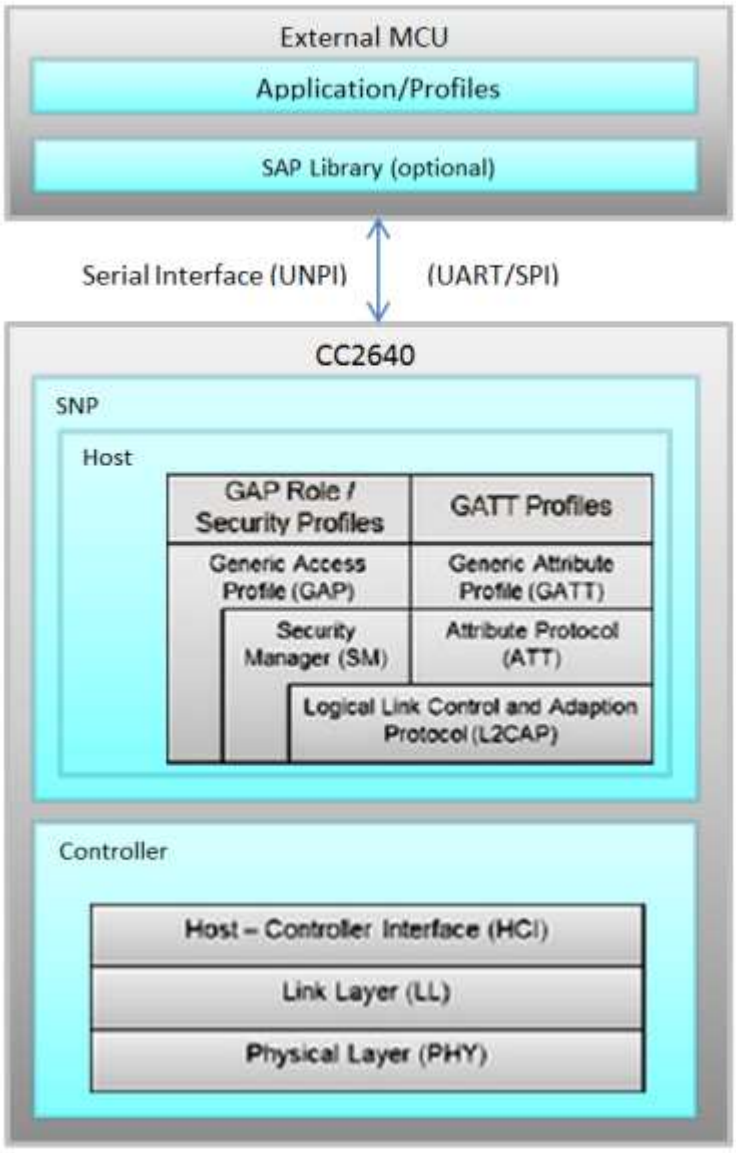

Рисунок 2.34 – Реалізація взаємодії основного контролеру та BLE контролеру

З боку мережевого контролера необхідно реалізувати розпакування пакетів і взаємодія з HUB рівнем. Склад системи розробки під мережевий контролер представлений на рисунку 2.35:

Рисунок 2.35 – Склад системи розробки під CC2650

Основні компоненти системи розробки:

1) Операційна система в режимі реального часу (RTOS) з ядром TI-RTOS SYS / BIOS
2) CC26xxware driverLib забезпечує рівень абстракції регістрів і використовується програмним забезпеченням та драйверами для CC2650 SoC
3) Стек протоколів низької енергії Bluetooth надається у вигляді бібліотеки з частинами стеку протоколів в ПЗУ CC2650.

Зразки додатків та профілів полегшують початок розробки, використовуючи як власні, так і загальні рішення. Основний шаблон проекту містить вихідний код як для стеку протоколу, так і для прикладного рівню. Склад шаблону проекту зображено на рисунку 2.36:

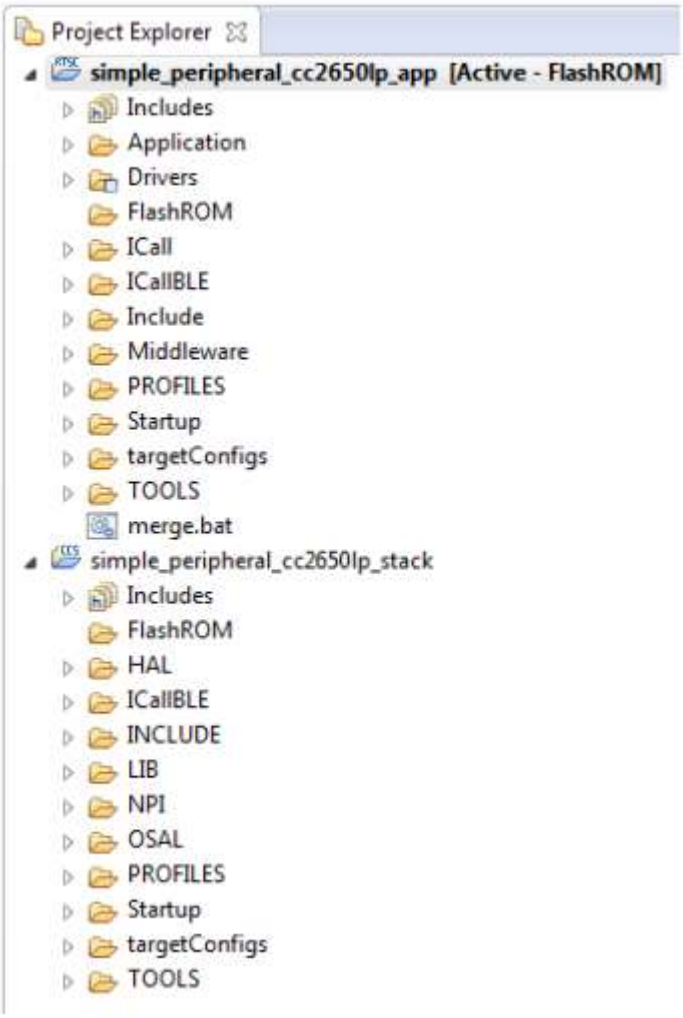

Рисунок 2.36 – Склад шаблону проекту

Проект стека включає нижні шари стека протоколу BLE а також включаючи рівні GAP та GATT. Більшість інших виробників надає код стеку як бібліотеку. Проект прикладного рівня включає профілі, код програми, драйвери та модуль ICall.

ICall_init () ініціалізує екземпляр модуля ICall, а ICall_createRemoteTasks () створює завдання на зовнішнє зображення з функцією введення за відомою адресою. Після ініціалізації ICall, завдання програми реєструється в ICall через ICall_registerApp. Після запуску планувальника SYS / BIOS і додаток Завдання виконується, програма надсилає команду протоколу, визначену в ICallBLEAPI.c, таку як GAP_GetParamValue(). Команда протоколу не виконується в потоці програми, а інкапсульована в повідомлення ICall і направлено до завдання стеку протоколу через фреймворк ICall. Ця команда надсилається диспетчеру ICall, де вона відправляється та виконується на на стороні сервера (тобто стек низької енергії Bluetooth). Потік програми тим часом блокується (тобто чекає) для відповідного повідомлення про стан команди (тобто статус і значення параметра GAP). Коли стек протоколів низького енергоспоживання Bluetooth завершує виконання команди, повідомлення про стан команди відповідь надсилається через ICall назад у потік програми. Приклад взаємодії представлений на рисунку 2.37:

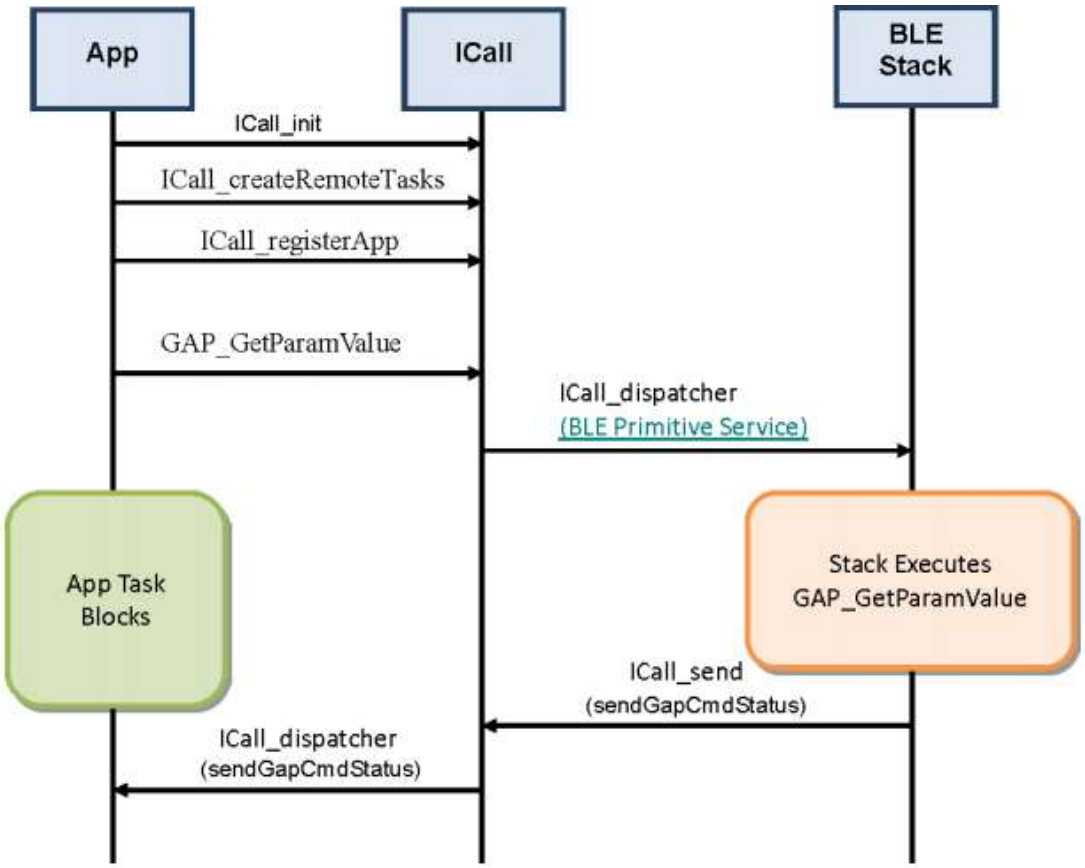

Рисунок 2.37 – Використання інтерфейсу ICall

Сам додаток обробляє пакети від основного контролера при в окремому потоці. Хід виконання потоку представлений на рисунку 2.38:

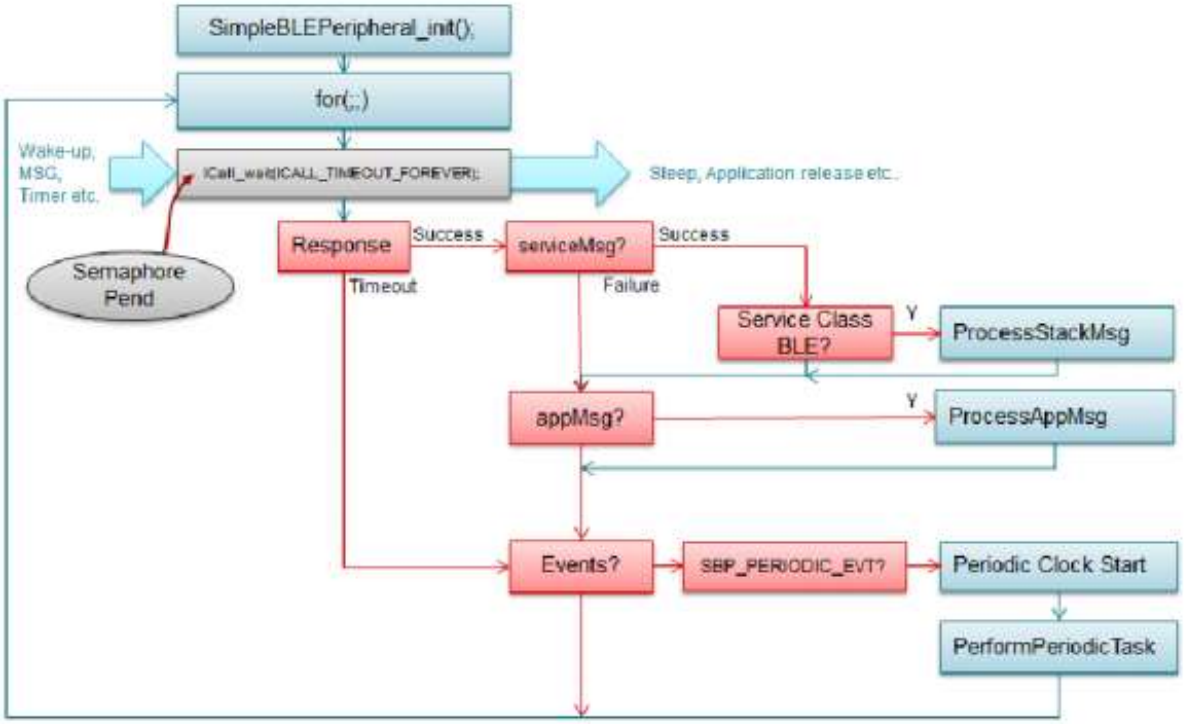

Рисунок 2.38 – Алгоритм обробки потоків мережевим контролером

Постачальник STK розробив рівень GATT стека протоколу Bluetooth Low Energy для використання додатком для даних зв'язок між двома підключеними пристроями. Дані передаються та зберігаються у формі характеристики, які зберігаються в пам'яті на пристрої Bluetooth Low Energy. У GATT коли два пристрої підключені, кожен виконує одну з двох ролей:

- Сервер GATT - Цей пристрій містить характерну базу даних, яку читає або записує GATT клієнт.
- Клієнт GATT - цей пристрій зчитує або записує дані із сервера GATT або на нього.

На це показано взаємозв'язок у зразку з'єднання Bluetooth Low Energy, де периферійний пристрій. Реалізація взаємодії продемонстровано на малюнку 2.39:

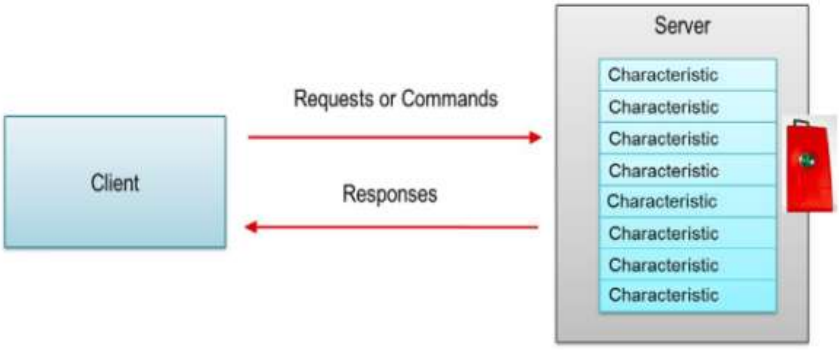

Рисунок 2.39– Взаємодія з стороннім пристроєм

Хоча характеристики іноді взаємозамінні, коли йдеться про Bluetooth Low Energy, враховуйте їх як групи інформації, що називаються атрибутами. Атрибути - це основні групи переданої інформації між пристроями. Характеристики організовують і використовують атрибути як значення даних, властивості та конфігурацію.
Типова характеристика складається з таких атрибутів:

• Значення характеристики: Це значення - це значення даних характеристики.
• Опис користувача характеристики: Цей опис являє собою рядок ASCII, що описує характеристику. Ці атрибути зберігаються на сервері GATT у таблиці атрибутів. Наступні властивості пов'язані з кожним атрибутом:
• Дескриптор - Ця властивість є індексом атрибута в таблиці. Кожен атрибут має унікальний дескриптор.
• Тип - Цей атрибут вказує, що представляють дані атрибута. Цей атрибут називається універсальним унікальний ідентифікатор (UUID). Деякі з цих UUID визначаються Bluetooth SIG, а інші визначаються користувачем.

2.6 Висновок з розділу 2

При виконанні розділу два було спроектовано архітектуру інформаційної системи проведення вібраційної діагностики. Після розроблення архітектури були реалізовані наступні елементи системи:

- Датчик та вбудоване ПЗ датчику;
- ПЗ проміжного рівня, що реалізує мережеву взаємодію;
- Хмарний сервіс обробки та зберігання даних;
- Інтерфейс оператора.

Також спроектовано та виготовлено стенд для перевірки функціоналу системи проведення вібромоніторінгу обладнання.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТУ І ФОРМУЛЮВАННЯ ПРАКТИЧНИХ РЕКОМЕНДАЦІЙ

3.1 Проведення експерименту оцінки функціоналу калібрування

Оцінювання роботи системи потрібно с перевірки функціоналу калібрування датчика. Для цього можливо використовувати відладку при використанні інтерфейсу JTAG. У датчика три осі і з цієї причини доцільно проводити перевірку калібрування шляхом установки окремої осі перпендикулярно площині робочого столу. Якщо кут витриманий вірно, показання для осі в такому випадку повинні становити значення сели тяжіння землі. Схема розташування осей зображена на на рисунку 3.1:

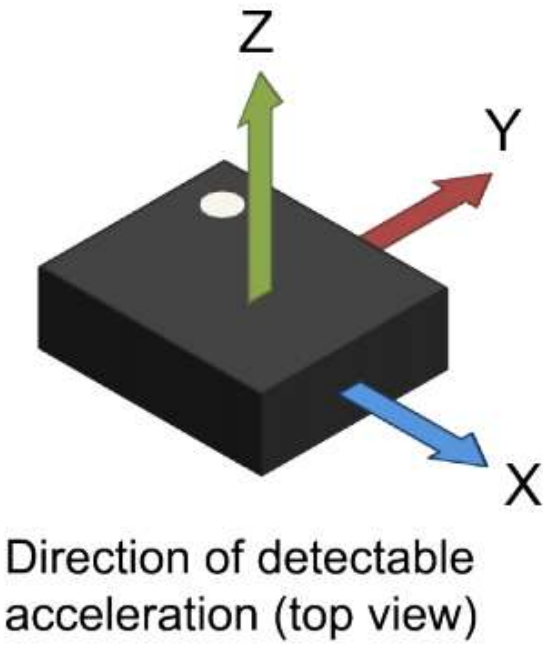

Рисунок 3.1 – Схематичне представлення направлень вимірювання

Результат вимірювать представлено на рисунку 3.2:

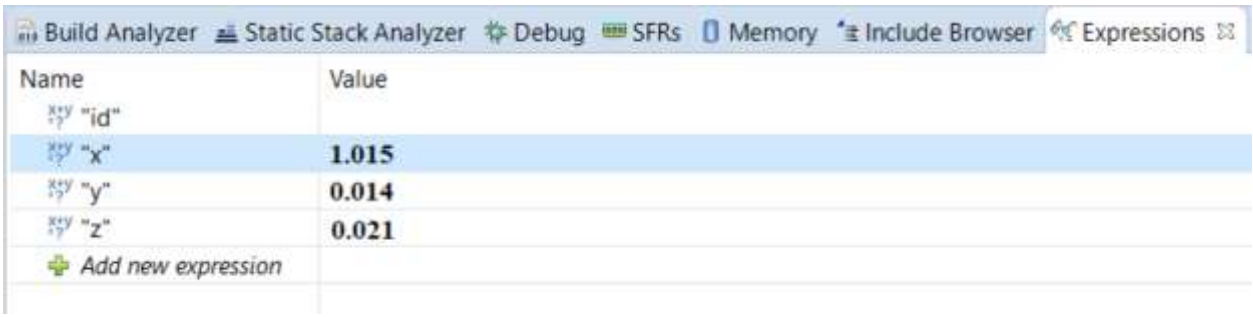

Рисунок 3.2 –Результат вимірювання

Аналогічні результати отримані з іншими осями. Після отримання результатів потрібно перевірити функціональність акселерометра за допомогою розробленого стенду.

3.2 Проведення експерименту за допомогою стенду

Для тесту оберти стенду виставлені на 190 оборотів в секунду. Результати отримані за допомогою відладчика і візуалізовано засобами відладчика середовища розробки. Результати представлено на рисунку 3.3:

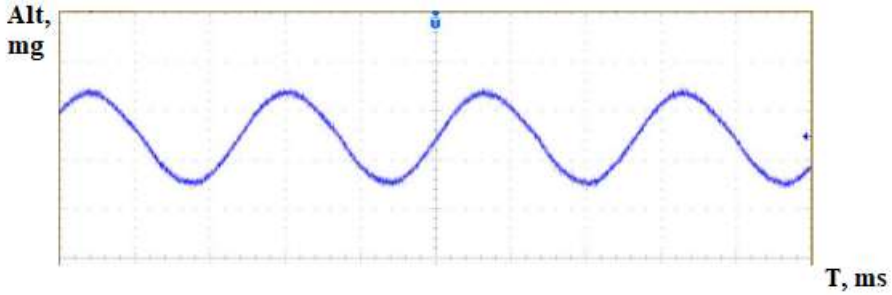

Рисунок 3.3 –Результат випробування стенду за допомогою відладчика

Для перевірки коректності результатів необхідно скопіювати отримані значення в ексель і провести перетворення в частотну область. Результат приведення представлений на рисунку 3.4:

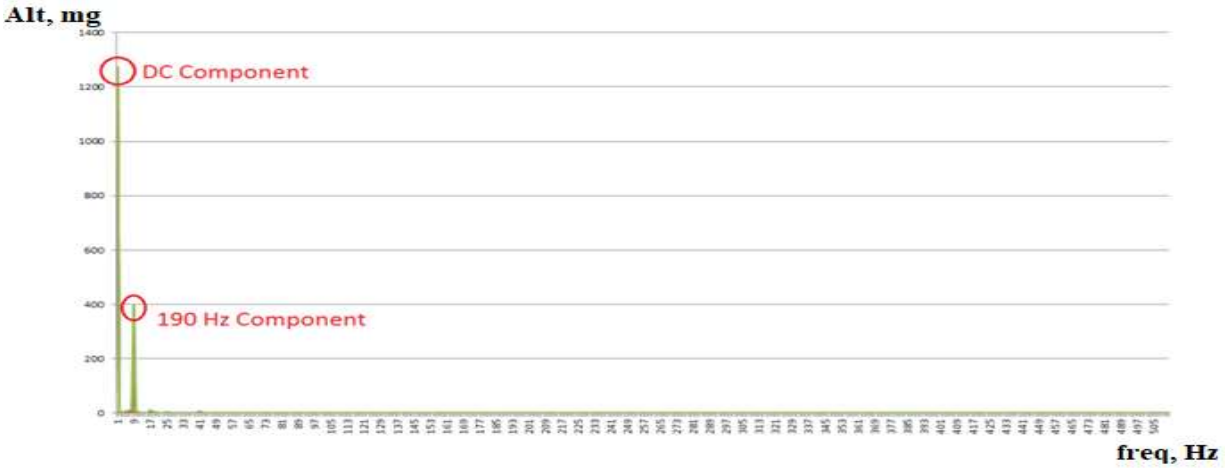

Рисунок 3.4 –Результат випробування стенду за допомогою відладчика після ШПФ

Наступним кроком є проведення інтеграційного тестування. Для реалізації інтеграційного тестування біло реалізовано додаток для Android, завдяки якому смартфон може працювати в якості HUB шлюзу.

Інтерфейс дуже простий, функціонал програми дозволяє лише підписатися на повідомлення джерела даних. Вибір проводитися за допомогою GUID.

Інтерфейс представлений на рисунку 3.5:

Рисунок 3.5 – Інтерфейс реалізації Hub рівня

Після вибору пристрою смартфон упакує повідомлення від BLE і передає в хмарний сервіс. Далі показники будуть візуалізовано за допомогою реалізованості програмного забезпечення. Результат візуалізації представлений на рисунку 3.6:

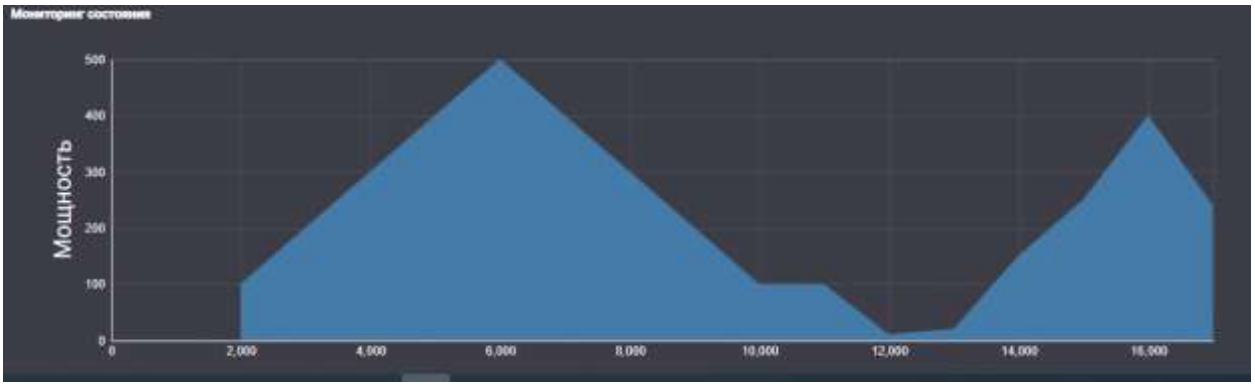

Рисунок 3.6 – Результати візуалізації спектру за допомогою хмарного рішення.

Також потрібно провести оцінку енергоспоживання використовуючи інструментальний інструментальні засоби, описані у розділі 2. Згідно виконаним вимірам, активна фаза датчику становить 70 мс, середній ток в активній фазі до 10 міліампер. Ток у фазі сну складає приблизно 5 мікроампер.

Для періоду вимірювань в 1 час середній ток складе 31 мікроампер. Для інтервалу 10 хвилин середній ток складе 300 мікроампер. При використанні акумулятору з ємністю 2500 міліампер(напр. типу CR123) можливо розраховувати на автономність у 11 місяців.

3.3 Оцінка результатів експерименту

По результатам експерименту розроблена система продемонструвала відповідність виділеним функціональним вимогам. Для оцінки актуальності роботи доцільно привести порівняння з аналогами.

3.3.1 Порівняння розробленої системи з аналогами

Для порівняння було обрано аналогічні три аналогічні системи, доступні на комерційному ринку. Було обрано системи виробників FLUKE, Baltech та Vi Block. Fluke та Vi Block розробляють системи стаціонарного бездротового моніторингу, Baltech пропонує рішення у вигляді портативного прибору. Приклад зовнішнього вигляду апаратної частини системи FLUKE представлено на рисунку 3.7:

Рисунок 3.7 – Апаратна частина системи вібромоніторінгу FLUKE

Архітектура системи FLUKE містить проміжний рівень у вигляді точки доступу та систему бездротових датчиків. Програмне забезпечення дозволяє оновлювати показник максимальної амплітуди вібрації раз в 120 секунд. Для передачі значень використовується протокол BLE 4.1. Інтерфейс програмного забезпечення представлено на рисунку 3.8:

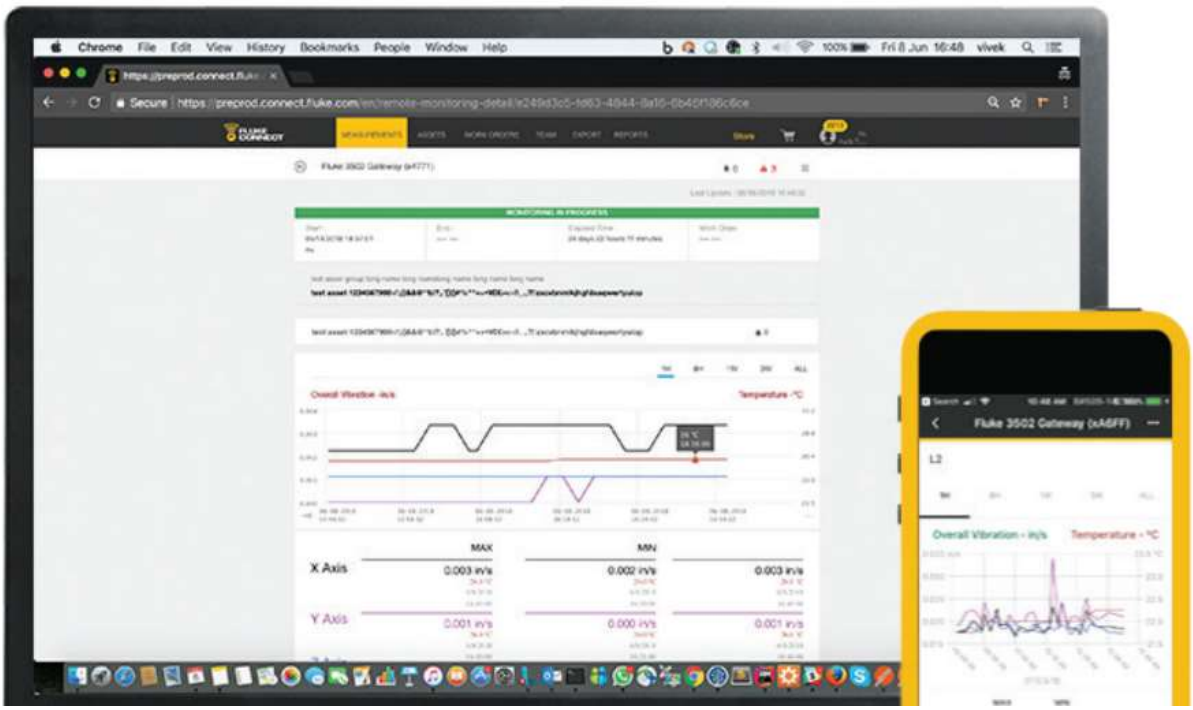

Рисунок 3.8 – Зовнішній вигляд ПЗ FLUKE

Архітектура системи ViBlock схожа на архітектуру рішення Fluke. Програмне забезпечення дозволяє ViBlock показник максимальної амплітуди вібрації раз в 60 секунд.

Для передачі значень використовується протокол BLE 4.2. Зовнішній вигляд апаратного забезпечення продемонстровано на рисунку 3.9:

Рисунок 3.9 – Зовнішній вигляд датчику Vi Block

Прикладне програмне забезпечення ViBlock надається у вигляді локального Windows додатку. Для інтеграції у систему керування підприємством потрібно самостійно реалізовувати сервіси та проміжні рівні доставки показників.

Baltech розробляє автономні прибори, інформації про можливість інтеграції з інформаційною системою підприємства немає.

Порівняльна характеристика апаратних можливостей систем представлена у таблиці 3.1:

Таблиця 3.1 – Порівняння апаратної частини

	FLUKE 3561	ViBlock	BALTECH CSI 9420	Розроблена система
Частотний діапазон, Гц	10-1000	10-1000	2-1000	10-6000
Бездротове рішення	Так	Так	Так	Так
Протокол бездротового зв'язку	BLE 4.1	BLE 4.2	IEC 62591	BLE 5.1
Мінімальний період опитування, сек	120	60	60	~1, для повного спектру 15
Інтеграція у інформаційну систему	Так, є API	Стандартного API немає	Так, є API	Так, є API
Розміри, мм	61.5 x 24	116x62	?	50x24
Час роботи без заміни батареї	До 3 років	До 5 років	?	До 1 року
Тип вихідних даних	Макс значення та частота	Макс значення та частота	Повний спектр	Повний спектр

Порівняльна характеристика програмних можливостей систем представлена у таблиці 3.2:

Таблиця 3.2 – Порівняння програмної частини

	FLUKE 3561	ViBlock	BALTECH CSI 9420	Розроблена система
Веб додаток	Так	Ні	Ні	Так
Мобільний додаток та оперативні повідомлення	Так	Ні	Ні	Так
Експертна система діагностики	Ні	Ні	Так	Ні

3.4 Висновки за третім розділом

В ході аналізу результатів експерименту було оцінено роботу системи вібраційної діагностики та сформовані практичні рекомендації по використанню системи. Перевірено функціонал калібрування датчику, отримання та передачі даних. Випробування проводились за допомогою розробленого стенду. Були виділені наступні переваги системи:

1) Частотний діапазон датчика ширший ніж у аналогів;
2) Можливість передачі повного діапазону спектру;
3) Сучасний стандарт бездротової передачі та наявність мобільного додатку;

ВИСНОВКИ

В результаті виконання роботи були виконанні наступні завдання:

1) проведено оцінку методів стану обладнання та аналіз наявних технічних рішень діагностики обладнання;
2) розроблена система вібраційної діагностики та оцінки стану обладнання;
3) виконано планування експерименту для дослідження системи вібраційної діагностики обладнання;
4) була виконана оцінка придатності до використання системи вібраційної діагностики та визначені її переваги;
5) сформовані практичні рекомендації застосування розроблених методів та програмно технічних рішень для оцінки обладнання за допомогою вібраційної діагностики.

В рамках виконання вищенаведених завдань спроектовано та реалізовано фізичну модель системи, яка включає мікроконтролер, трьохосьовий цифровий акселерометр IIS3DWB та характеризується низькою ціною технічного рішення.

Розроблено та реалізовано спеціалізоване ПЗ системи, яке включає драйвер для налаштування, збору і опрацювання даних з акселерометра та відповідне ПЗ для побудови графіків сигналів віброприскорення в часовій і частотній областях. Побудоване ПЗ дає змогу реалізувати широкі функціональні можливості та є вільновикористовуваним.

Побудована система дає можливість проводити аналіз параметрів вібрації з метою передбачення і запобігання можливих аварій, зменшуючи таким чином затрати пов’язані з виходом із ладу ріжучого інструмента, дорогих деталей і вузлів обертового обладнання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rață, G. System for Monitoring and Analysis of Vibrations at Electric Motors [Text] / G. Rață, M. Rață // Intern. Journal of Emerging Technology and Advanced Engineering. – 2014. – Vol. XXI, Issue 3. – P. 97–104.

2. Milovančević, M. Embedded Systems for Vibration Monitoring [Text] / M. Milovančević, A. Veg, A. Makedonski, J. Stefanović Marinović // Facta Universitatis. Series: Mechanical Engineering. – 2014. – Vol. 12, Issue 2. – P. 171–181.

3. Rocha, S. M. S. Method to Measure Displacement and Velocity from Acceleration Signals [Text] / S. M. S. Rocha, J. F. S. Feiteira, P. S. N. Mendes, U. P. B. Da Silva, R. F. Pereira // Intern. Journal of Engineering Research and Applications. – 2016. – Vol. 6, Issue 6. – P. 52–59.

4. Sekiya, H. Technique for Determining Bridge Displacement Response Using MEMS Accelerometers [Text] / H. Sekiya, K. Kimura, C. Miki // Sensors. – 2016. – Vol. 16, Issue 2. – P. 257. doi: 10.3390/s16020257

5. Goyal, D. Development of non-contact structural health monitoring system for machine tools [Text] / D. Goyal, B. S. Pabla // Journal of Applied Research and Technology. – 2016. – Vol. 14, Issue 4. – P. 245–258. doi: 10.1016/j.jart.2016.06.003

6. Albarbar, A. Suitability of MEMS Accelerometers for Condition Monitoring: An experimental study [Text] / A. Albarbar, S. Mekid, A. Starr, R. Pietruszkiewicz // Sensors. – 2008. – Vol. 8, Issue 2. – P. 784–799. doi: 10.3390/s8020784

7. Hjort, A. Measuring mechanical vibrations using an Arduino as a slave I/O to an EPICS control system [Text] / A. Hjort, M. Holmberg. – Uppsala University, 2015. – 25 p. Not a reprint

10. Chaudary, S. B. Vibration Monitoring of Rotating Machines Using MEMS Accelerometer [Text] / S. B. Chaudary, M. Sengupta, K. Mukherjee // Intern. Journal of Sceintific Engineering and Research. – 2014. – Vol. 2, Issue 9.

ДОДАТОК А

А.1 Лістинг коду

Пз датчику – main.c файл

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "..\..\iis3_drv\iis3.h"
#define ARM_MATH_CM4
#include "stdio.h"
#include "arm_const_structs.h"
#include "arm_math.h"
#include "stm32l4xx_ll_exti.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define SHOW_FFT_DATA                                   // Uncomment to print
FFT PSD data on the serial monitor
//#define NORMALIZE PSD                                 // Uncomment to print
PSD normalized to the largest bin

#define FIFO_WATERMARK      256

#define FFT_LENGTH          2048                        // FFT length must be a
power of 2! Allowed values are: 16, 32, 64, 128, 256, 512
#define SCAN_AVG_COUNT      1

// Analyze Ax, Ay, Az or Arms; uncomment one only
//#define ANALYZE_AX
//#define ANALYZE_AY
//#define ANALYZE_AZ
#define ANALYZE_RMS
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART_UART_Init(void);
/* USER CODE BEGIN PFP */
void fft_process(float32_t * p_input, const arm_cfft_instance_f32 * p_input_struct,float32_t * p_output,
uint16_t output_size);
void transmitString(char* str);
void transmitFrameEnd();
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t id = 0;
uint32_t              start_fft;                                    // Time immediately before calling
FFT analysis
uint32_t               finish_fft;                                   // Time immediately after
completing FFT analysis
float32_t              fft_io_buffer[2*FFT_LENGTH];                  // Complex FFT
input/output buffer
float32_t              fft_output_buffer[FFT_LENGTH];                // Real magnitude
output buffer
float32_t              psd_f32[FFT_LENGTH];                          // Scan-averaged
complex magnitude PSD output buffer
const arm_cfft_instance_f32* fft_Instance;                           // CMSIS f32 CFFT
instance pointer for the selected FFT length
uint16_t               fft_sample_count = 0;                         // Index to count the number
of input samples in the fft I/O buffer
float                  freq_bin_width   = 26667.0f/2.0f;             // FFT Frequency bin
width in Hz
float                 psd_max         = 0.0f;                       // PSD maximum for spectrum
normalization
volatile uint8_t       fft_data_ready   = 0;                         // Flag for when FFT raw
data buffer is full and ready for analysis
uint8_t                psd_avg_count    = 0;                         // FFT scan counter for scan
averaging
uint32_t               m_ifft_flag      = 0;                         // Flag that selects forward (0)
or inverse (1) CFFT transform.
uint32_t               m_do_bit_reverse = 1;

float aRes,ax,ay,az;

uint8_t Ascale = AFS_4G;
char TXbuff[40] = {0};
float accelBias[3] = {0.0f, 0.0f, 0.0f};
int16_t IIS3DWBData[4] = {0};
uint8_t IIS3DWB_DataReady = 0, IIS3DWB_Wakeup = 1;

uint16_t fifo_count;                                 // FIFO buffer size variable                  // create
FIFO data array
float Ax[FFT_LENGTH], Ay[FFT_LENGTH], Az[FFT_LENGTH];  // create accel data buffers
uint8_t fifo_mode = Contmode;
uint32_t start_dataRead, stop_dataRead;
/* USER CODE END 0 */
```

```
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */


  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */


  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_SYSCFG);
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

  NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* System interrupt init*/

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_USART_UART_Init();
  /* USER CODE BEGIN 2 */
  LL_mDelay(10);

        reset();
    id = readByte(0x0F);
    printf("Who am I result: 0x%x\r\n", id);
    LL_mDelay(2);

    aRes = getAres(Ascale);
    TestResult res = selfTest(aRes);

    printf("\r\n");
    printf("Accel Self Test:\r\n");
    printf("+Ax result: %f mg\r\n",res.Axresults);
    printf("-Ax result: %f mg\r\n",res.NAxresults);
    printf("+Ay result: %f mg\r\n",res.Ayresults);
    printf("-Ay result: %f mg\r\n",res.NAyresults);
    printf("+Az result: %f mg\r\n",res.Azresults);
    printf("-Az result: %f mg\r\n",res.NAzresults);
```

```
  printf("Should be between 90 and 1700 mg\r\n");
  printf("\r\n");

  init(Ascale);
  offsetBias(accelBias,aRes);

  switch(FFT_LENGTH)                                    // Define the appropriate
CMSIS f32 CFFT instance pointer for the defined FFT length
    {
     case 16:
       fft_Instance = &arm_cfft_sR_f32_len16;
       break;
     case 32:
       fft_Instance = &arm_cfft_sR_f32_len32;
       break;
     case 64:
       fft_Instance = &arm_cfft_sR_f32_len64;
       break;
     case 128:
       fft_Instance = &arm_cfft_sR_f32_len128;
       break;
     case 256:
       fft_Instance = &arm_cfft_sR_f32_len256;
       break;
     case 512:
       fft_Instance = &arm_cfft_sR_f32_len512;
       break;
     case 1024:
       fft_Instance = &arm_cfft_sR_f32_len1024;
       break;
     case 2048:
       fft_Instance = &arm_cfft_sR_f32_len2048;
       break;
     case 4096:
       fft_Instance = &arm_cfft_sR_f32_len4096;
       break;
     default:
       //printf("Incorrect FFT length specified!");
       while(1) {;}
       break;
    }

     freq_bin_width /= (float)FFT_LENGTH;
    //freq_bin_width/=2;
     memset(fft_io_buffer, 0x00, sizeof(fft_io_buffer));

    //LL_EXTI_EnableIT_0_31(LL_EXTI_LINE_11);
    LL_EXTI_EnableIT_0_31(LL_EXTI_LINE_10);
    LL_mDelay(2);
    uint16_t i = 0;
  /* USER CODE END 2 */



  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

```
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
  //  LL_mDelay(500 );
    if(IIS3DWB_DataReady)  // Handle data ready condition
        {
         IIS3DWB_DataReady = 0;
//       IIS3DWBstatus = IIS3DWB.DRstatus(); // read data ready status
//       if (IIS3DWBstatus & 0x01) {         // if new accel data is available, read it

         readAccelData(IIS3DWBData);

           if(i<2049)
           {
          // Now we'll calculate the accleration value into actual g's
                Ax[i] = (float)IIS3DWBData[0]*aRes - accelBias[0];   // get actual g value, this depends on scale being set
             Ay[i] = (float)IIS3DWBData[1]*aRes - accelBias[1];
             Az[i] = (float)IIS3DWBData[2]*aRes - accelBias[2];
             i++;
           } else
           {
                for(uint8_t j = 0; j<FFT_LENGTH/FIFO_WATERMARK;j++)
                {
                        for (uint16_t ii = 0; ii < 256; ii++)
                                {

                                 // Now we'll load the new acceleration data into the FFT I/O buffer
                                #if defined(ANALYZE_AX)

fft_io_buffer[2*fft_sample_count+2*ii] = Ax[ii];
                                #elif defined(ANALYZE_AY)

fft_io_buffer[2*fft_sample_count+2*ii] = Ay[ii];
                                #elif defined(ANALYZE_AZ)

fft_io_buffer[2*fft_sample_count+2*ii] = Az[ii];
                                #elif defined(ANALYZE_RMS)

fft_io_buffer[2*fft_sample_count+2*ii] = sqrtf(Ax[ii]*Ax[ii] + Ay[ii]*Ay[ii] + Az[ii]*Az[ii]);
                                #else

fft_io_buffer[2*fft_sample_count+2*ii] = Ax[ii];                 // Default to ax in case the analysis definition is botched
                                #endif

fft_io_buffer[2*fft_sample_count+2*ii+1] =0.0f;                         // Pad the imaginary component with 0.0f
                                 }
                        fft_sample_count += 256;
                }
                fft_sample_count = 0;
                fft_data_ready   = 1;
                i=0;
```

```
                    }
//      }
    }
//if (IIS3DWB_Wakeup) { // if activity change event FALLING detected
  //       IIS3DWB_Wakeup = 0;

        // printf("IIS3DWB is awake\r\n");
  //   } // end activity c

if(fft_data_ready)                                          // If the I/O buffer
has a full FFT sample in it, build the FFT input buffer and calculate the FFT
 {
  fft_data_ready = 0;                                       // Reset the FFT
data ready flag                                     // Diagnostic to track FFT calculation time; comment
out when done
  fft_process(fft_io_buffer,      fft_Instance,      fft_output_buffer,      FFT_LENGTH);
// Calculate the PSD!                                       // Diagnostic to track FFT calculation time;
comment out when done
  for(uint32_t i=0; i<FFT_LENGTH/2; i++)                    //
Add new PSD values to the avrages array element-by-element
  {
   psd_f32[i] += fft_output_buffer[i];
  }
  psd_avg_count++;                                          // Increment
the scan averaging counter
 }

if(psd_avg_count >= SCAN_AVG_COUNT)                         //
If the requested number of PSD samples have been summed, calculate averages and report the PSD over serial
{
  psd_avg_count = 0;                                        // Reset the scan
averaging counter flag
  psd_max      = 0.0f;
  for(uint32_t i=0; i<FFT_LENGTH/2; i++)                    // The
number of PSD bins is half the FFT length
  {
   psd_f32[i] /= (float)SCAN_AVG_COUNT;                     //
Divide PSD sum arrray element by the number of scans
   psd_f32[i] *= 2.0f;                                      // Multiply by 2;
symmetrical about the Nyquist frequency
   if(psd_f32[i] > psd_max) {psd_max = psd_f32[i];}         //
Track maximum amplitude bin as we go...
  }
  for(uint32_t i=0; i<FFT_LENGTH/2; i++)                    // The
number of PSD bins is half the FFT length
  {
   #ifdef SHOW_FFT_DATA
    sprintf(TXbuff,"1,%.1f,%.2f\r\n",(float)i*freq_bin_width                +
freq_bin_width/2.0f,psd_f32[i]);
    transmitString(TXbuff);// Calculate the center frequency of each PSD bin
    LL_mDelay(10);
    //void transmitFrameEnd();
    #ifdef NORMALIZE PSD
        printf("%.5f | ",psd_f32[i]/psd_max);               // Print PSD data
normalized to the largest bin                       // Print raw PSD data
    #endif
   #endif
```

```
                                psd_f32[i] = 0.0f;                                  // Clear PSD sum
array element after you print it so the psd_f32[] will be cleared and ready for the next sample
                                }
                                LL_mDelay(500);
                                transmitFrameEnd();
                                #ifdef SHOW_FFT_DATA
                                #endif

                               }
                        }
  /* USER CODE END 3 */
 }

 /**
  * @brief System Clock Configuration
  * @retval None
  */
 void SystemClock_Config(void)
 {
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_2);

   if(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_2)
  {
  Error_Handler();
  }
  LL_PWR_SetRegulVoltageScaling(LL_PWR_REGU_VOLTAGE_SCALE1);
  LL_RCC_MSI_Enable();

   /* Wait till MSI is ready */
  while(LL_RCC_MSI_IsReady() != 1)
  {

  }
  LL_RCC_MSI_EnableRangeSelection();
  LL_RCC_MSI_SetRange(LL_RCC_MSIRANGE_11);
  LL_RCC_MSI_SetCalibTrimming(0);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_MSI);

   /* Wait till System clock is ready */
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_MSI)
  {

  }
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
  LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);

  LL_Init1msTick(48000000);

  LL_SYSTICK_SetClkSource(LL_SYSTICK_CLKSOURCE_HCLK);
  LL_SetSystemCoreClock(48000000);
  LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_PCLK2);
 }

 /**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
```

```
 */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  LL_SPI_InitTypeDef SPI_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_SPI1);

  LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA);
  /**SPI1 GPIO Configuration
  PA5   ------> SPI1_SCK
  PA6   ------> SPI1_MISO
  PA7   ------> SPI1_MOSI
  */
  GPIO_InitStruct.Pin = LL_GPIO_PIN_5|LL_GPIO_PIN_6|LL_GPIO_PIN_7;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_5;
  LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  SPI_InitStruct.TransferDirection = LL_SPI_FULL_DUPLEX;
  SPI_InitStruct.Mode = LL_SPI_MODE_MASTER;
  SPI_InitStruct.DataWidth = LL_SPI_DATAWIDTH_8BIT;
  SPI_InitStruct.ClockPolarity = LL_SPI_POLARITY_HIGH;
  SPI_InitStruct.ClockPhase = LL_SPI_PHASE_2EDGE;
  SPI_InitStruct.NSS = LL_SPI_NSS_SOFT;
  SPI_InitStruct.BaudRate = LL_SPI_BAUDRATEPRESCALER_DIV8;
  SPI_InitStruct.BitOrder = LL_SPI_MSB_FIRST;
  SPI_InitStruct.CRCCalculation = LL_SPI_CRCCALCULATION_DISABLE;
  SPI_InitStruct.CRCPoly = 7;
  LL_SPI_Init(SPI1, &SPI_InitStruct);
  LL_SPI_SetStandard(SPI1, LL_SPI_PROTOCOL_MOTOROLA);
  LL_SPI_SetRxFIFOThreshold(SPI1,LL_SPI_RX_FIFO_TH_QUARTER);
  LL_SPI_DisableNSSPulseMgt(SPI1);
  /* USER CODE BEGIN SPI1_Init 2 */
  LL_SPI_Enable(SPI1);
  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
```

```
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

        LL_LPUART_InitTypeDef LPUART_InitStruct = {0};

        LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

        /* Peripheral clock enable */
        LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_LPUART1);

        LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOC);
        /**LPUART1 GPIO Configuration
        PC0   ------> LPUART1_RX
        PC1   ------> LPUART1_TX
        */
        GPIO_InitStruct.Pin = LL_GPIO_PIN_0|LL_GPIO_PIN_1;
        GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
        GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
        GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
        GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
        GPIO_InitStruct.Alternate = LL_GPIO_AF_8;
        LL_GPIO_Init(GPIOC, &GPIO_InitStruct);

        /* USER CODE BEGIN LPUART1_Init 1 */

        /* USER CODE END LPUART1_Init 1 */
        LPUART_InitStruct.BaudRate = 115200;
        LPUART_InitStruct.DataWidth = LL_LPUART_DATAWIDTH_8B;
        LPUART_InitStruct.StopBits = LL_LPUART_STOPBITS_1;
        LPUART_InitStruct.Parity = LL_LPUART_PARITY_NONE;
        LPUART_InitStruct.TransferDirection = LL_LPUART_DIRECTION_TX_RX;
        LPUART_InitStruct.HardwareFlowControl = LL_LPUART_HWCONTROL_NONE;
        LL_LPUART_Init(LPUART1, &LPUART_InitStruct);
        LL_LPUART_EnableDMADeactOnRxErr(LPUART1);
        LL_LPUART_Enable(LPUART1);
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART_UART_Init(void)
{

       /* USER CODE BEGIN USART1_Init 0 */

        /* USER CODE END USART1_Init 0 */

        LL_USART_InitTypeDef USART_InitStruct = {0};
```

```
    LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Peripheral clock enable */
    LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_USART1);

    LL_AHB1_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA);
    /**USART1 GPIO Configuration
    PA9   ------> USART1_TX
    PB7   ------> USART1_RX
    */
    GPIO_InitStruct.Pin = LL_GPIO_PIN_9;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
    LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
    GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
    GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
    GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
    GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
    LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    /* USER CODE BEGIN USART1_Init 1 */

    /* USER CODE END USART1_Init 1 */
    USART_InitStruct.BaudRate = 115200;
    USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
    USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
    USART_InitStruct.Parity = LL_USART_PARITY_NONE;
    USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
    USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
    USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
    LL_USART_Init(USART1, &USART_InitStruct);
    LL_USART_ConfigAsyncMode(USART1);
    LL_USART_Enable(USART1);
    /* USER CODE BEGIN USART1_Init 2 */

    /* USER CODE END USART1_Init 2 */

}
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA);
  LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOC);
```

```
  LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOB);

  /**/
  LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_4);

  /**/
  GPIO_InitStruct.Pin = LL_GPIO_PIN_4;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOC, &GPIO_InitStruct);


  GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
   GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
   GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
   LL_GPIO_Init(GPIOB, &GPIO_InitStruct);


  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTB, LL_SYSCFG_EXTI_LINE10);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_10;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);


  /* EXTI interrupt init*/
  NVIC_SetPriority(EXTI15_10_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),3, 0));
  NVIC_EnableIRQ(EXTI15_10_IRQn);
 }

 /* USER CODE BEGIN 4 */
 void fft_process(float32_t * p_input, const arm_cfft_instance_f32 * p_input_struct,                // Calculate
complex FFT and complex magnitude using sp float CMSIS DSP library calls
                 float32_t * p_output, uint16_t output_size)
 {
   arm_cfft_f32(p_input_struct, p_input, m_ifft_flag, m_do_bit_reverse);                          // Use 32bit
float CFFT module to process the data
   arm_cmplx_mag_f32(p_input, p_output, output_size);                                             // Calculate the
magnitude at each bin using Complex Magnitude Module function
 }

 void transmitString(char* str)
 {
        uint16_t i = 0;
        while (i<strlen(str))
         {
                while (!LL_LPUART_IsActiveFlag_TXE(LPUART1)) {}
                LL_LPUART_TransmitData8(LPUART1, str[i]);

                while (!LL_USART_IsActiveFlag_TXE(USART1)) {}
                LL_USART_TransmitData8(USART1, str[i]);
```

```
                i++;
        }// Calculate the magnitude at each bin using Complex Magnitude Module function
}
void transmitFrameEnd()
{
        uint16_t i = 0;
        char str[2] = "end";
        while (i<3)
         {
                while (!LL_LPUART_IsActiveFlag_TXE(LPUART1)) {}
                LL_LPUART_TransmitData8(LPUART1, str[i]);

                while (!LL_USART_IsActiveFlag_TXE(USART1)) {}
                LL_USART_TransmitData8(USART1, str[i]);
                i++;
        }// Calculate the magnitude at each bin using Complex Magnitude Module function
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(char *file, uint32_t line)
{

    tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
}
#endif /* USE_FULL_ASSERT */
```

Файл iis3.c

```
#include "iis3.h"


uint8_t readByte(uint8_t adr)
{
        LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_4);
         uint8_t res = 0;
         while(!LL_SPI_IsActiveFlag_TXE(SPI1)) {}
```

```
	LL_SPI_TransmitData8(SPI1, (adr & 0x7F)|0x80);
	while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
	LL_SPI_ReceiveData8(SPI1);
	while(LL_SPI_IsActiveFlag_BSY(SPI1)) {}
	LL_SPI_TransmitData8(SPI1, 0);
	while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
	res = LL_SPI_ReceiveData8(SPI1);
	uint8_t i = 30;
	while(i > 0) {i--;}
	LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_4);
	return res;
}

void readBytes(uint8_t adr, uint8_t count, uint8_t * dest)
{
	LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_4);
		while(!LL_SPI_IsActiveFlag_TXE(SPI1)) {}
		LL_SPI_TransmitData8(SPI1, (adr & 0x7F)|0x80);
		while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
		LL_SPI_ReceiveData8(SPI1);

		 for (uint8_t ii = 0; ii < count; ii++)
		 {
			while(!LL_SPI_IsActiveFlag_TXE(SPI1)) {}
			LL_SPI_TransmitData8(SPI1, 0);
			while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
			dest[ii] = LL_SPI_ReceiveData8(SPI1);
			uint8_t i = 30;
			while(i > 0) {i--;}
		 }
	LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_4);
}

void writeByte(uint8_t adr, uint8_t value)
{
	LL_GPIO_ResetOutputPin(GPIOC, LL_GPIO_PIN_4);
		while(!LL_SPI_IsActiveFlag_TXE(SPI1)) {}
		LL_SPI_TransmitData8(SPI1, (adr & 0x7F));
		while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
		LL_SPI_ReceiveData8(SPI1);
		while(!LL_SPI_IsActiveFlag_TXE(SPI1)) {}
		LL_SPI_TransmitData8(SPI1, value);
		while(!LL_SPI_IsActiveFlag_RXNE(SPI1)) {}
		LL_SPI_ReceiveData8(SPI1);
		uint8_t i = 30;
		while(i > 0) {i--;}
	LL_GPIO_SetOutputPin(GPIOC, LL_GPIO_PIN_4);
}

void sleep()
{
 uint8_t temp = readByte(IIS3DWB_CTRL1_XL);
 writeByte(IIS3DWB_CTRL1_XL, temp & ~(0xA0));  // clear top three bits
}


void wake()
{
```

```
  uint8_t temp = readByte(IIS3DWB_CTRL1_XL);
  writeByte(IIS3DWB_CTRL1_XL, temp | 0xA0 );  // enable accelerometer
 }

 void reset()
 {
  writeByte(IIS3DWB_CTRL1_XL, 0x00); // set accel to power down mode
  uint8_t temp = readByte(IIS3DWB_CTRL3_C);
  writeByte(IIS3DWB_CTRL3_C, temp|0x01); // Set bit 0 to 1 to reset IIS3DWB
  LL_mDelay(1); // Wait for all registers to reset
 }

 uint8_t DRstatus()
 {
  uint8_t temp = readByte(IIS3DWB_STATUS_REG); // read data ready status register
  return temp;
 }

 uint8_t ACTstatus()
 {
  uint8_t temp = readByte(IIS3DWB_ALL_INT_SRC); // read activity status register
  return temp;
 }

 void init(uint8_t Ascale)
 {
  writeByte(IIS3DWB_INT1_CTRL, 0x01);          // enable data ready interrupt on INT1
  writeByte(IIS3DWB_COUNTER_BDR_REG1, 0x80);   // enable pulsed (not latched) data ready interrupt

  // enable block update (bit 6 = 1), auto-increment registers (bit 2 = 1)
  writeByte(IIS3DWB_CTRL3_C, 0x40 | 0x04);
  // by default, interrupts active HIGH, push pull
  // (can be changed by writing to bits 5 and 4, resp to above register)

  //  mask data ready until filter settle complete (bit 3 == 1), disable I2C (bit 2 == 1)
  writeByte(IIS3DWB_CTRL4_C, 0x08 | 0x04);

  writeByte(IIS3DWB_CTRL1_XL, 0xA0 | Ascale << 2); // set accel full scale and enable accel

  // activity interrupt handling
  writeByte(IIS3DWB_WAKE_UP_DUR, 0x08);       // set inactivity duration at 1 LSB = 512/26.667 kHz
ODR (so about 0.15 seconds)
  writeByte(IIS3DWB_WAKE_UP_THS, 0x02);       // set wake threshold to 62.5 mg at 4 G FS, so 4G/2^6
= 0.0625 G
  // (change SLOPE_EN to 0x00 to drive activity change to INT)
  writeByte(IIS3DWB_SLOPE_EN, 0x20);          // drive activity status to interrupt
  writeByte(IIS3DWB_INTERRUPTS_EN, 0x80);     // enable wakeup and activity/inactivity logic
  writeByte(IIS3DWB_MD2_CFG, 0x80);           // route activity change event to INT2
 }

 TestResult selfTest(float  _aRes)
 {
  int16_t temp[3] = {0, 0, 0};
  int16_t accelPTest[3] = {0, 0, 0}, accelNTest[3] = {0, 0, 0};
  int16_t accelNom[3] = {0, 0, 0};

  writeByte(IIS3DWB_CTRL1_XL, 0xA0 | AFS_4G << 2); // set accel full scale and enable accel
```

```
  writeByte(IIS3DWB_CTRL3_C, 0x40 | 0x04);          // enable block update (bit 6 = 1), auto-increment registers (bit 2 = 1)
  LL_mDelay(100); // wait 100 ms for stable function

  readAccelData(temp); // discard first sample
  LL_mDelay(100);
  readAccelData(temp); // should average these five times, but once is good enough
  accelNom[0] = temp[0];
  accelNom[1] = temp[1];
  accelNom[2] = temp[2];

  writeByte(IIS3DWB_CTRL5_C, 0x01); // positive accel self test
  LL_mDelay(100); // let accel respond
  readAccelData(temp); // discard first sample
  LL_mDelay(100);
  readAccelData(temp);
  accelPTest[0] = temp[0];
  accelPTest[1] = temp[1];
  accelPTest[2] = temp[2];

  writeByte(IIS3DWB_CTRL5_C, 0x02); // negative accel self test
  LL_mDelay(100); // let accel respond
  readAccelData(temp); // discard first sample
  LL_mDelay(100);
  readAccelData(temp);
  accelNTest[0] = temp[0];
  accelNTest[1] = temp[1];
  accelNTest[2] = temp[1];

  writeByte(IIS3DWB_CTRL5_C, 0x00); // normal mode
  writeByte(IIS3DWB_CTRL1_XL, 0x00); // power down accel and report results

  TestResult res = {};

  res.Axresults = ((accelPTest[0] - accelNom[0]) * _aRes * 1000.0);
  res.NAxresults =((accelNTest[0] - accelNom[0]) * _aRes * 1000.0);
  res.Ayresults = ((accelPTest[1] - accelNom[1]) * _aRes * 1000.0);
  res.NAyresults = ((accelNTest[1] - accelNom[1]) * _aRes * 1000.0);
  res.Azresults = ((accelPTest[2] - accelNom[2]) * _aRes * 1000.0);
  res.NAzresults = ((accelNTest[2] - accelNom[2]) * _aRes * 1000.0);
  return res;
  LL_mDelay(2000);
 }

 void readAccelData(int16_t * destination)
 {
  uint8_t rawData[6];  // x/y/z accel register data stored here
  readBytes(IIS3DWB_OUTX_L_XL, 6, &rawData[0]);  // Read the 6 raw accel data registers into data array
  destination[0] = (int16_t)((int16_t)rawData[1] << 8)  | rawData[0] ;  // Turn the MSB and LSB into a signed 16-bit value
  destination[1] = (int16_t)((int16_t)rawData[3] << 8)  | rawData[2] ;
  destination[2] = (int16_t)((int16_t)rawData[5] << 8)  | rawData[4] ;
 }

 int16_t readTempData()
 {
  uint8_t rawData[2];  // x/y/z accel register data stored here
```

```
  readBytes(IIS3DWB_OUT_TEMP_L, 2, &rawData[0]);  // Read the 2 raw temperature data registers into data array
  int16_t temp = (int16_t)((int16_t)rawData[1] << 8)  | rawData[0] ;  // Turn the MSB and LSB into a signed 16-bit value
  return temp;
}

void offsetBias(float * destination, float _aRes)
{
  int16_t temp[3] = {0, 0, 0};
  int32_t sum[3] = {0, 0, 0};

  LL_mDelay(10000);

  for (uint8_t ii = 0; ii < 128; ii++)
  {
    readAccelData(temp);
    sum[0] += temp[0];
    sum[1] += temp[1];
    sum[2] += temp[2];
    LL_mDelay(10);
  }

  destination[0] = sum[0] * _aRes / 128.0f;
  destination[1] = sum[1] * _aRes / 128.0f;
  destination[2] = sum[2] * _aRes / 128.0f;

  if (destination[0] > 0.75f)  {
    destination[0] -= 1.0f; // Remove gravity from the x-axis accelerometer bias calculation
  }
  if (destination[0] < -0.75f) {
    destination[0] += 1.0f; // Remove gravity from the x-axis accelerometer bias calculation
  }
  if (destination[1] > 0.75f)  {
    destination[1] -= 1.0f; // Remove gravity from the y-axis accelerometer bias calculation
  }
  if (destination[1] < -0.75f) {
    destination[1] += 1.0f; // Remove gravity from the y-axis accelerometer bias calculation
  }
  if (destination[2] > 0.75f)  {
    destination[2] -= 1.0f; // Remove gravity from the z-axis accelerometer bias calculation
  }
  if (destination[2] < -0.75f) {
    destination[2] += 1.0f; // Remove gravity from the z-axis accelerometer bias calculation
  }
}

float getAres(uint8_t Ascale) {
        float _aRes = 0;
  switch (Ascale)
  {
    // Possible accelerometer scales (and their register bit settings) are:
    // 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
    case AFS_2G:
      _aRes = 2.0f / 32768.0f;
      return _aRes;
      break;
    case AFS_4G:
```

```
      _aRes = 4.0f / 32768.0f;
      return _aRes;
      break;
    case AFS_8G:
      _aRes = 8.0f / 32768.0f;
      return _aRes;
      break;
    case AFS_16G:
      _aRes = 16.0f / 32768.0f;
      return _aRes;
      break;
  }
}

void initFIFO(uint16_t fifo_size, uint8_t fifo_mode)
{
// FIFO size if 9 bits here stored in a 16-bit uint16_t, maximum is 0x01FF = 511
  writeByte(IIS3DWB_FIFO_CTRL1, (fifo_size & 0x00FF));       // write lowest 8 bits
  writeByte(IIS3DWB_FIFO_CTRL2, 0x80 | (fifo_size & 0x0100) >> 8); // write highest of 9 bits to bit 0,
enable stop on watermark (bit 7 == 1)
  writeByte(IIS3DWB_FIFO_CTRL3, 0x0A);                    // write to FIFO at 26667 Hz
// time stamp (bits 6-7 == 0) and temperature (bits 4 - 5 == 0) not stored in FIFO
  writeByte(IIS3DWB_FIFO_CTRL4, fifo_mode);                // select FIFO mode
  writeByte(IIS3DWB_INT1_CTRL, 0x08);                      // enable FIFO threshold interrupt on INT1
}

uint16_t FIFOstatus()
{
  uint8_t temp1 = readByte(IIS3DWB_FIFO_STATUS1); // read FIFO status 1 register
  uint8_t temp2 = readByte(IIS3DWB_FIFO_STATUS2); // read FIFO status 2 register
  if(temp2 & 0x80) // if watermark full bit asserted, then FIFO ready to be read
  {
    uint16_t FIFOcount = (uint16_t) ((temp2 & 0x03) << 8) | temp1; // overwrite count with FIFO size if
watermark interrupt asserted
    return FIFOcount;
  }
}

//void readFIFOData(uint16_t fifo_count, int16_t * xyzData)
//{
//  static uint8_t rawData[1 + 512*7]; // maximum FIFO size
 // rawData[0] = IIS3DWB_FIFO_DATA_OUT_TAG | 0x80;
 // SPI.beginTransaction(SPISettings(IIS3DWB_SPI_CLOCK, MSBFIRST, IIS3DWB_SPI_MODE));
  //digitalWrite(_cs, LOW);
  //SPI.transfer(rawData, (1 + (fifo_count * 7)));
  //digitalWrite(_cs, HIGH);
  //SPI.endTransaction();
 // for(uint32_t ii = 1; ii < (uint32_t)(1 + (fifo_count * 7)); ii += 7)
 // {
  //*xyzData++ = (rawData[ii+2] << 8) | rawData[ii+1];
 // *xyzData++ = (rawData[ii+4] << 8) | rawData[ii+3];
 // *xyzData++ = (rawData[ii+6] << 8) | rawData[ii+5];
  //}
//}
```