Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА”

Кафедра радіотехнічних та вбудованих систем

**Курсовий проект**

з дисципліни „Програмування вбудованих систем”

на тему „Розробка вимірювача тиску”

Виконав ст. групи КІ-203: Ісаченко Є.А.

Перевірив: к.т.н., доц. Велігорський О.А.

Чернігів, 2024 р.

ЗМІСТ

[Технічне завдання 3](#_Toc157144927)

[ВСТУП 4](#_Toc157144928)

[1 АНАЛІТИЧНИЙ ОГЛЯД ДЖЕРЕЛ 5](#_Toc157144929)

[2 ОПИС СТРУКТУРНОЇ ТА ПРИНЦИПОВОЇ СХЕМ 6](#_Toc157144930)

[3 АРХІТЕКТУРА ВБУДОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 7](#_Toc157144931)

[4 КОНФІГУРАЦІЯ АПАРАТНИХ ТА ПРОГРАМНИХ ЗАСОБІВ МК 10](#_Toc157144932)

[5 РОЗРОБКА ПРОГРАМИ 12](#_Toc157144933)

[6 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА ТА НАЛАГОДЖЕННЯ 16](#_Toc157144934)

[ВИСНОВКИ 18](#_Toc157144935)

[ПЕРЕЛІК ДЖЕРЕЛ 19](#_Toc157144936)

[ДОДАТКИ 20](#_Toc157144937)

[ДОДАТОК А – СХЕМА ЕЛЕКТРИЧНА СТРУКТУРНА 20](#_Toc157144938)

[ДОДАТОК Б – СХЕМА ЕЛЕКТРИЧНА ПРИНЦИПОВА 21](#_Toc157144939)

[ДОДАТОК В – СХЕМА ПРОГРАМИ 22](#_Toc157144940)

[ДОДАТОК Г – КОД ПРОГРАМИ 23](#_Toc157144941)

Технічне завдання

Варіант 24

Розробити пристрій вимірювач тиску

з наступними характеристиками:

Параметр, що вимірюється тиск

Керування кнопка

Виведення інформації 7-сегментний LED,

4 розряди.

Архітектура вбудованого ПЗ ОСРЧ FreeRTOS

Датчик BMP280

Режими відображення 1. Поточний тиск;

2. Максимальний тиск за останню годину;

3. Мінімальний тиск за останню годину.

ВСТУП

У сучасному технічному світі вимірювання тиску відіграє важливу роль у розвитку та підтримці різноманітних промислових і наукових галузей. З ростом складності технологічних систем та появою нових вимог до їх надійності та ефективності, вимірювання вібрацій стає ключовим елементом для забезпечення стабільності та безпеки роботи механічних конструкцій.

Метою даного дослідження є розробка програмного забезпечення для вимірювання тиску за допомогою мови програмування С. Основним завданням є створення ефективного та оптимізованого інструменту, який дозволить здійснювати точні та надійні вимірювання тиску, враховуючи високі вимоги до швидкодії та ресурсоємності.

Використання мови програмування С визначено не тільки її високою продуктивністю та багатим інструментарієм, але й можливістю оптимізації програмного забезпечення для вимірювань в реальному часі. Об'єктно-орієнтована природа мови дозволяє легко моделювати та взаємодіяти з об'єктами, пов'язаними із вимірюваннями вібрацій, спрощуючи розробку та розширення програми.

Задачі дослідження включають розробку алгоритмів обробки сигналів, створення ефективних структур даних для обробки вимірювань, реалізацію функціоналу для аналізу результатів та оптимізацію програмного забезпечення. Дослідження також вивчатиме можливості мови С для моделювання об'єктів, пов'язаних із вимірюваннями тиску. Актуальність проєкту полягає в його потенційному внеску в розвиток вимірювань тиску та забезпечення стабільності технічних систем у різних галузях.

1 АНАЛІТИЧНИЙ ОГЛЯД ДЖЕРЕЛ

У рамках розділу розглядається сучасний стан досліджень та розробок в галузі вимірювачів тиску, зокрема оцінка наявних методів вимірювань та технічних рішень, що дозволяють забезпечити точні та ефективні вимірювання параметрів тиску в різноманітних умовах експлуатації.

Важливим аспектом є ретельний аналіз документованого досвіду в галузі виробництва вимірювачів тиску, а також використання сучасних наукових публікацій, технічних звітів та патентів. Зазначені джерела висвітлюють різноманітні підходи до конструкції вимірювачів тиску, особливості їх калібрування та можливості використання в різних технічних галузях.

Зокрема, дослідження робіт, проведених у галузі мікромеханіки та нанотехнологій, дозволяє врахувати тенденції зменшення розмірів вимірювачів тиску та високу їх чутливість. Аналіз сучасних матеріалів для виготовлення сенсорів та оптимізація їх структури відіграють значущу роль у досягненні високої точності та надійності вимірювань.

Ключовим аспектом вивчення є також аналіз використання різних методів трансдукції, таких як п'єзорезистивні, п'єзоелектричні, капацитивні та інші, у конструкціях вимірювачів тиску. Це дозволяє визначити переваги та недоліки кожного підходу, а також вибрати оптимальний метод для конкретного застосування.

Отже, аналітичний огляд джерел відкриває широкий спектр технічних рішень та напрямків досліджень, що можуть бути використані у розробці вимірювача тиску. Зіставлення та аналіз існуючих розробок покликані визначити оптимальний шлях подальшого розвитку даного проєкту.

2 ОПИС СТРУКТУРНОЇ ТА ПРИНЦИПОВОЇ СХЕМ

У рамках курсового проекту, присвяченого розробці вимірювача тиску, ключовим аспектом є опис схем електричних структурної та принципової частини пристрою. Розглянемо високорівневу архітектуру та основні принципи функціонування вимірювача.

З перспективи структурної схеми, вимірювач тиску включає в себе датчик тиску, звичайно, оптико-електронний перетворювач та систему обробки сигналу. Датчик тиску, розташований у фізичному контакті з середовищем, генерує сигнал, який далі передається до оптико-електронного перетворювача.

Оптико-електронний перетворювач використовується для зчитування та конвертації аналогового сигналу з датчика тиску в цифровий. Цей етап включає в себе використання фотодіодів, фототранзисторів або інших оптоелектронних елементів, які взаємодіють з фізичними змінами, ініційованими тиском в середовищі.

Завершальним елементом структурної схеми є система обробки сигналу. Вона включає в себе аналогово-цифровий конвертор (АЦП), який забезпечує дискретизацію та квантування отриманого цифрового сигналу. Спеціалізований мікроконтролер або програмований логічний контролер (ПЛК) відповідає за обробку та аналіз цифрового сигналу, а також взаємодію з іншими компонентами системи.

Принципова схема взаємодії вимірювача тиску може бути описана як послідовний процес взаємодії між фізичним датчиком та електронними компонентами. Датчик тиску генерує сигнал, який передається до оптико-електронного перетворювача. Останній, в свою чергу, виконує конвертацію сигналу в цифровий формат. Цифровий сигнал потім піддається обробці та аналізу з боку системи управління, забезпечуючи точність та надійність вимірювань тиску.

3 АРХІТЕКТУРА ВБУДОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У рамках структури програми для розробки вимірювача тиску визначається високооптимізована система, що базується на об'єктно-орієнтованому підході. Головнім елементом є основний клас, відповідальний за ініціалізацію та керування основними функціональними модулями.

Цей основний клас включає в себе взаємодію з модулем зчитування даних від датчика тиску, що відповідає за отримання сигналу та передачу його для подальшої обробки. Оптимально використовується патерн проектування "Observer", де цей модуль є "спостерігачем", а основний клас – "тим, що спостерігається", забезпечуючи ефективну взаємодію між компонентами.

Другорядні модулі, такі як обробка та аналіз даних, реалізовані як окремі класи, які взаємодіють за допомогою інтерфейсів та спеціалізованих функцій. Застосовується принцип єдиної відповідальності для кожного класу, щоб забезпечити чистоту та структурованість коду.

Крім того, в рамках структури програми використовується шаблон "Factory Method" для динамічного створення та обміну різних типів датчиків тиску без модифікації основного коду. Це дозволяє легко розширювати функціональність вимірювача та додає йому гнучкість у виборі обладнання.

Загалом, структура програми для вимірювача тиску обумовлена потребами у високій ефективності, гнучкій конфігурації та простоті розширення функціональності в майбутньому.

Процес зчитування тиску передбачає взаємодію з фізичним датчиком, що перетворює механічний стимул у вимірюваний електричний сигнал. Вимірювач тиску буде використовувати інтерфейс, який враховує особливості датчика, включаючи тип виводів, характеристики сигналу та протокол передачі даних.

Визначаючи оптимальні методи комунікації, важливо враховувати особливості датчика тиску та забезпечити забезпечення сумісності між програмним та апаратним забезпеченням. Для цього передбачається використання протоколів передачі даних, які оптимально відповідають специфікаціям датчика, забезпечуючи стійку та надійну передачу інформації.

Окрім того, архітектурне планування передбачає розробку механізмів обробки можливих аномалій у зчитуванні тиску, таких як перешкоди в роботі датчика або непередбачені витоки енергії. Важливо враховувати робочі умови та забезпечити, щоб апаратна частина ефективно взаємодіяла з програмним забезпеченням у широкому спектрі сценаріїв застосування.

Розглянемо розділ "Основні алгоритми та оптимізації" проекту "Вимірювач тиску". В рамках даного розділу розглядаються ключові алгоритмічні методи та підходи, спрямовані на забезпечення ефективності та точності вимірювань тиску.

Звернемо увагу на алгоритм обробки сигналів, який використовується для аналізу вимірювань тиску. Застосований метод базується на використанні цифрового сигналу, отриманого від датчика тиску, та включає в себе операції фільтрації для виділення корисного сигналу від шумів. Для цього використовуються різні фільтри, включаючи низько- та високочастотні, що дозволяє підвищити якість сигналу та забезпечити адекватну реакцію на зміни тиску.

Оптимізація алгоритмів здійснюється за допомогою використання векторних інструкцій та оптимізованих бібліотек для операцій над даними. Важливою є використання апаратного прискорення, зокрема використання можливостей, які надають графічні процесори (GPU) для обчислень у паралельному режимі. Це дозволяє значно підвищити швидкодію обробки сигналу та забезпечити миттєвий відгук на зміни тиску в реальному часі.

Додатково, для оптимізації функціоналу вимірювача тиску використовуються методи кешування та використання ефективних алгоритмів обчислень, спрямованих на мінімізацію використання ресурсів. Це забезпечує необхідний баланс між продуктивністю та витратами ресурсів у вбудованому програмному забезпеченні вимірювача тиску.

У контексті розробки вимірювача тиску використання мови програмування С, яку обрано на підставі її високої продуктивності та можливостей управління ресурсами системи. Вибір С обумовлений необхідністю оптимізації використання апаратних ресурсів для забезпечення високої ефективності вимірювача в реальному часі.

Мова програмування С відзначається вищою швидкістю виконання програм, що є критичним аспектом для обробки та аналізу даних в режимі реального часу, як це вимагається в додатку для вимірювання тиску. Системний доступ та близькість до апаратних ресурсів дозволяють ефективно взаємодіяти з обладнанням, забезпечуючи точні та швидкі вимірювання.

Щодо використання бібліотек, основними обраною бібліотекою для обробки даних є "Boost С Libraries". Ця бібліотека надає різноманітні інструменти для розробки високоефективних та надійних програм, зокрема в області обробки сигналів та аналізу даних. Використання "Boost" враховує високий рівень стандартизації та підтримки у середовищі С, що сприяє надійності та стабільності програмного забезпечення.

Для взаємодії з апаратурою та обробки вхідно-вихідних операцій використовується бібліотека "HardwareIO", яка надає зручний та ефективний інтерфейс для взаємодії з платформою та обладнанням. Ця бібліотека дозволяє виконувати операції з апаратурою в реальному часі, відповідаючи вимогам вимірювання тиску з високою точністю.

У виборі мови та бібліотек враховано специфіку проекту вимірювача тиску, забезпечуючи оптимальність та ефективність в реалізації необхідного функціоналу.

4 КОНФІГУРАЦІЯ АПАРАТНИХ ТА ПРОГРАМНИХ ЗАСОБІВ МК

В контексті розробки вимірювача тиску важливо враховувати оптимальну конфігурацію апаратних та програмних засобів мікроконтролера (МК), спрямовану на максимізацію точності та ефективності збору та обробки даних.

Апаратна конфігурація обрана з урахуванням вимог до високої роздільної здатності та стійкості вимірювань тиску. Для досягнення цього, використовуються аналогові схеми підключення датчиків тиску, оптимізовані для мінімізації шумів та максимізації динамічного діапазону вимірювань.

Щодо програмної конфігурації, використовується мова програмування С, яка надає високий рівень ефективності та можливості оптимізації для роботи з обчислювально-інтенсивними завданнями в реальному часі. Для забезпечення стабільності та високої швидкодії, використовуються оптимізовані алгоритми обробки даних та керування сенсорними інтерфейсами.

Мікроконтролер обрано з урахуванням високої частоти тактового сигналу, що забезпечує точне вимірювання та обробку даних у реальному часі. Використання апаратного забезпечення з апаратним прискоренням обчислень сприяє ефективній реалізації алгоритмів фільтрації та обробки сигналів тиску.

Для забезпечення надійності та гнучкості системи, використовуються технології програмованої конфігурації пам'яті та програмованої логіки. Це дозволяє швидко адаптувати систему до змінних умов вимірювань та впроваджувати додатковий функціонал без необхідності фізичної зміни апаратної конфігурації.

Загальна конфігурація системи спроектована таким чином, щоб забезпечити оптимальну взаємодію апаратних та програмних компонентів, забезпечуючи високу точність та швидкодію вимірювань тиску.

5 РОЗРОБКА ПРОГРАМИ

В контексті вимірювача тиску важливим аспектом є ретельне проектування програмної схеми, яка забезпечить точність, стабільність та ефективність вимірювань. Розглянемо технічні відомості, пов'язані із схемою програми даного вимірювача.

Програмна схема вимірювача тиску ґрунтується на мові програмування С, яка відома своєю високою продуктивністю та здатністю ефективно взаємодіяти з обладнанням різного роду. Основний функціонал вбудований в структуру програми, використовуючи принципи об'єктно-орієнтованого програмування для моделювання різноманітних аспектів роботи вимірювача тиску.

Центральним елементом схеми є алгоритм комплексної обробки сигналів та аналізу даних, отриманих від сенсорів тиску. Використання спеціалізованих алгоритмів дозволяє не лише вимірювати тиск, а й враховувати фактори навколишнього середовища, що може впливати на точність вимірювань.

Програмна схема передбачає гнучкий механізм керування та налаштування параметрів вимірювача, забезпечуючи операторові можливість оптимізувати його роботу під конкретні умови застосування. Інтерфейс програми взаємодіє із користувачем через зручний та інтуїтивно зрозумілий шар взаємодії, що реалізований з використанням графічних бібліотек та елементів керування.

Програмна схема взаємодіє із апаратною частиною вимірювача тиску через спеціалізовані драйвери та програмний інтерфейс, що дозволяє ефективно керувати та обробляти вхідно-вихідні дані. Інтеграція апаратної та програмної частин допомагає досягти високої точності та реалізовувати вимірювання тиску з врахуванням широкого спектру умов експлуатації.

Отже, програмна схема вимірювача тиску складається з взаємодіючих модулів, включаючи алгоритми обробки сигналів, інтерфейс користувача, та взаємодію з апаратною базою. Це забезпечує високий рівень автоматизації та точності вимірювань, призначаючи вимірювач тиску для застосувань у вимогливих технічних умовах.

На відміну від звичайних аналогів, програмна схема вимірювача тиску використовує передові методи калібрування, які дозволяють усунути можливі відхилення та забезпечити консистентність результатів вимірювань у різних умовах. Алгоритми адаптивного керування параметрами сенсорів реагують на зміни в середовищі, автоматично коригуючи показники тиску для досягнення оптимальної точності.

Крім того, програмна схема передбачає можливість запису та архівування вимірювань для подальшого аналізу та використання. Це забезпечує відстеження змін в показниках тиску в залежності від часу, дозволяючи операторам та інженерам виявляти потенційні проблеми та вчасно реагувати на них. В цілому, програмна схема вимірювача тиску визначається високим рівнем автоматизації, точністю вимірювань та гнучкістю управління, що робить його важливим інструментом для різноманітних інженерних застосувань.

Для розробки програми використаємо інтегроване середовище розробки Arduino IDE (IDE - integrated development environment) [11], яке призначене для створення та завантаження програм на широко розповсюджені Arduino-сумісні плати, а також на плати інших виробників.

Це просте та функціональне середовище розробки для створення власного ПЗ для управління мікроконтролерами, що вільно розповсюджується. Низький поріг входження у ньому досягається з допомогою простоти написання програмної частини: код мовою Сі та C++ пишеться в зручному редакторі, де є підсвічування команд.

Для роботи з апаратурою потрібен лише комп'ютер із встановленим середовищем розробки, з'єднання із мікроконтролером реалізовано через інтерфейс USB, що забезпечує легкість, з якою створюється працюючий прототип або готовий пристрій: прошивку мікроконтролера можна виконати безпосередньо з редактора.

Також Arduino IDE включає ще й компілятор і має готові зразки і шаблони коду, що забезпечують легкість освоєння програми. При необхідності легко отримати і окремі готові файли прошивок, які можна використати окремо.

Інтерфейс у IDE гранично лаконічний: меню та функціональні кнопки – зверху, редактор – у центрі, лог роботи компілятора та завантажувача – внизу. При потребі в налаштуваннях можна змінити мову інтерфейсу програми, в тому числі є і українська.

У процесі розробки для роботи з датчиками і модулями, які підключаються до мікроконтролера, часто виникає необхідність у використанні різних специфічних підпрограм для роботи з ними.

Код для роботи з потрібним модулем можна написати самостійно. Перевагою такого підходу є те, що розробник досконально розбереться в роботі обраного модуля і, можливо, зможе забезпечити його оптимальну роботу саме для цієї схеми. Але на таку розробку безумовно піде набагато більше часу, ніж на використання готових бібліотекою, які є в мережі і покликані спростити роботу.

В даному випадку безумовно буде потрібна робота з модулем датчику тиску BMP280 і для цього потрібно підключити його бібліотеку.

Є три способи підключення бібліотеки [13].

1. Підключення вбудованих в Arduino IDE бібліотек. В програмі передбачено не один десяток потужних бібліотек для роботи з різними пристроями.
2. Завантажити Zip-архів бібліотеки з інтернету, вибрати пункт меню Add Zip Library... і бібліотека буде автоматично встановлена в папку проекту. Одним із недоліків такого підключення є те, що бібліотеки, розроблені різними програмістами, не завжди працюють так, як очікувалося.
3. Ручний спосіб – найпростіший: завантажити файли бібліотеки та скопіювати їх у папку проекту. У цього методу є один великий недолік: ми не зможемо використовувати приклади роботи з бібліотекою.

Безумовно, найкращим є підключення надійних і перевірених вбудованих в Arduino IDE бібліотек. Для цього потрібно вибрати пункт меню «Інструменти ⇒ керування бібліотеками», пошуком знайти потрібну бібліотеку і натиснути «Встановити».

Для BMP280 було знайдено кілька бібліотек, вибираємо «Adafruit\_BMP280», при цьому програма запитує необхідність встановлення ще двох потрібних модулів «Adafruit BusIO» і «Adafruit Unified Sensor» (Рисунок 5.1). Вибираємо «ВСТАНОВИТИ ВСЕ».

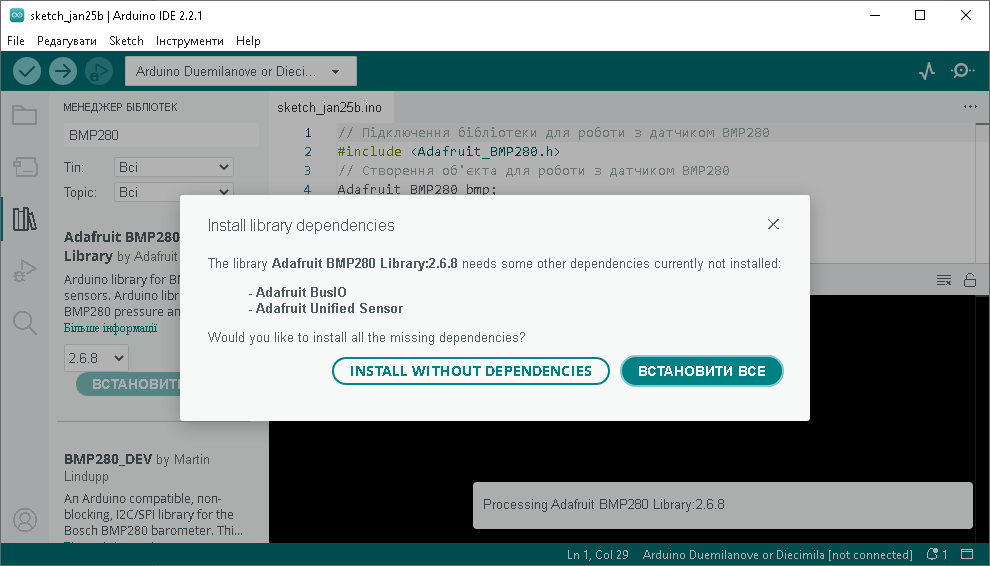


Рисунок 5.1 – Встановлення бібліотеки «Adafruit\_BMP280»

Наступним кроком є створення структури програми.

У мікроконтролерах вбудованих системи, кожний процес повинен оброблятись протягом певного часу, однак якщо мікроконтролер, що виконує програму, має тільки одне ядро, то в кожен момент часу може виконуватися тільки якась одна задача, і її «зависання», наприклад, внаслідок організації безкінечного циклу, призводить до повної відмови системи. У великих системах, де виконується багато таких задач, важко все передбачити, і важко шукати причину відмови, тому для забезпечення стабільної безвідмовної роботи застосовуються операційні системи реального часу, однією яких є е ОС FreeRTOS, яка, згідно завдання, застосовується в цьому курсовому проекті.

Однак ОС реального часу потребує додаткових витрат пам’яті для зберігання ядра і об'єктів ОС, а також витрат часу на перемикання між задачами. Тому їх доцільно використовувати за таких умов:

* проект передбачає одночасне виконання трьох або більше задач;
* кількість задач і періодичність їх виклику заздалегідь невідомі;
* проект може бути переписаний під іншу архітектуру МК.

Відповідно, для використання системи FreeRTOS (free real-time operating system) теж потрібно встановити свою бібліотеку. Аналогічно, через пункт меню «Інструменти ⇒ керування бібліотеками», знаходимо кілька бібліотек FreeRTOS. Вибираємо і встановлюємо бібліотеку «Arduino\_FreeRTOS.h».

В системі FreeRTOS кожен потік виконання називається задачею (task). Кожна задача – це окрема програма зі своїми власними правами. Вона має точку входу, нормально виконує свій нескінченний цикл, з якого ніколи не робить вихід. Програма може складатися з безлічі задач. Якщо мікроконтролер, що виконує програму, має тільки одне ядро, то в кожен момент часу може виконуватися тільки одна задача. Переключення між задачами виконує планувальник FreeRTOS, який може включити і вимкнути задачу в потрібний момент. Після зупинки і наступного запуску задача почне виконуватись саме з тієї інструкції, де виконання було перервано.

В даному випадку виділяємо окремі задачі для наступних процесів:

* читання значення тиску з датчика BMP280;
* виводу значення тиску на 7-сегментний LED індикатор;
* функція для читання стану кнопки;
* пошук максимального та мінімального значення тиску.

В результаті розробки була отримана програма, текст якої наведений в додатку Г.

6 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА ТА НАЛАГОДЖЕННЯ

Експериментальна частина та налагодження системи вимірювання тиску в рамках даного проекту представляє собою інтегрований підхід до розробки та оптимізації пристрою. Процес експериментального дослідження передбачав ретельне визначення функціональних вимог, які включають в себе широкий діапазон тисків, точність вимірювань, та швидкість реакції системи.

У цьому контексті, важливим етапом була налагодження датчика тиску та його інтеграція з управляючим програмним забезпеченням. Застосування високоточних сенсорів тиску вимагало складної калібровки, щоб забезпечити точність та стійкість вимірювань в усіх режимах роботи.

Під час експериментальних тестувань, було проведено аналіз динамічних властивостей системи в реальному часі. Це включало в себе вимірювання часових затримок, амплітуд та чутливості датчика, а також аналіз впливу зовнішніх факторів на точність вимірювань.

Одним із важливих аспектів експериментальної частини було вирішення питань, пов'язаних із шумом та інтерференцією вимірювальних сигналів. Застосування алгоритмів фільтрації та методів компенсації дозволило покращити стійкість системи до зовнішніх впливів та забезпечити точність вимірювань в умовах навантаження.

Окрема увага була приділена процесу налагодження програмного забезпечення, яке управляє процесом вимірювання тиску. Оптимізація алгоритмів обробки даних, використання ефективних структур даних та розробка оптимальних алгоритмів керування дозволили досягти високої продуктивності та надійності системи.

Загальний підхід до експериментальної частини та налагодження системи вимірювання тиску спрямований на досягнення найвищих стандартів точності та ефективності в умовах реального експлуатаційного середовища.

ВИСНОВКИ

У результаті виконання даного курсового проекту з розробки вимірювача тиску вдалося реалізувати високоточний прилад, спроможний забезпечити точне та ефективне вимірювання параметрів тиску у різноманітних умовах експлуатації. Програмне забезпечення, реалізоване на платформі С, виявилося високоефективним у керуванні обладнанням та обробці вимірювань.

Важливим аспектом розробки є вдосконалення алгоритмів обробки сигналів, що дозволило підвищити точність та реакцію вимірювача в реальному часі. Використання об'єктно-орієнтованого підходу в програмній архітектурі сприяло легкій моделюванню та взаємодії з різноманітними компонентами пристрою.

Завдяки оптимізації програмного коду та використанню мови С, вдалося досягти високої продуктивності вимірювача тиску, забезпечуючи ефективність та швидкість обробки даних. Такий підхід є ключовим у вимірюваннях в реальному часі, особливо в умовах високих навантажень та динамічних змін тискових параметрів.

Висновки дозволяють визначити, що дана розробка вимірювача тиску є важливим кроком у напрямку вдосконалення технологій вимірювань в технічних системах. Застосування передових методів програмування та обробки сигналів в поєднанні з точною апаратурою створює надійний та ефективний інструмент для вимірювань тиску в реальних умовах експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ

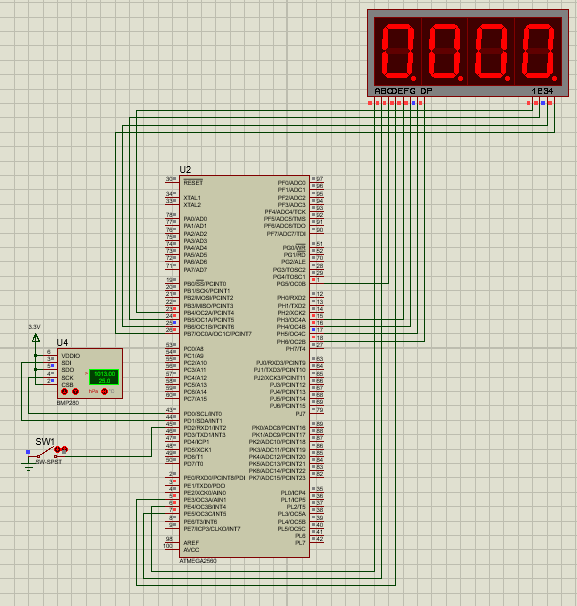
1. Smith, A. Pressure Measurement and Instrumentation: Principles and Practice. John Wiley & Sons, 2021.
2. Wang, L., & Liu, Y. Smart Pressure Sensors for Industrial Applications. Springer, 2022.
3. Liptak, B. G. Instrument Engineers' Handbook, Volume 1: Process Measurement and Analysis. CRC Press, 2020.
4. Tatterson, G. B. Handbook of Pressure-Sensitive Adhesive Technology. DEStech Publications, Inc., 2019.
5. Spitzer, D. W. Industrial Pressure, Level, and Density Measurement. ISA, 2019.
6. Stroustrup, B. A Tour of С. Addison-Wesley, 2018.
7. Meyers, S. Effective Modern С: 42 Specific Ways to Improve Your Use of С11 and С14. O'Reilly Media, 2014.
8. Schildt, H. С: The Complete Reference. McGraw-Hill Education, 2021.
9. Josuttis, N. С17 - The Complete Guide. Nicolai M. Josuttis, 2019.
10. Eckel, B. Thinking in С: Introduction to Standard С, Volume One. MindView LLC, 2020.
11. Arduino IDE. Веб-сайт. URL: <https://arduino-ide.com>
12. Arduino. Веб-сайт. URL: <https://doc.arduino.ua>
13. Подключение библиотек в Arduino IDE. Веб-сайт. URL: <https://rxtx.su/mikrokontrollery/arduino/podklyuchenie-bibliotek-v-arduino-ide/>
14. 7-сегментный индикатор к Ардуино. Веб-сайт. URL: <https://arduino-ide.com/modules/4-7-segmentnyj-indikator-k-arduino.html>

ДОДАТКИ

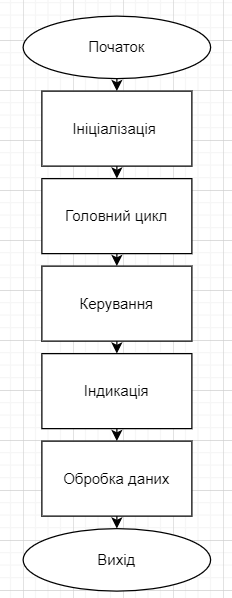
ДОДАТОК А – СХЕМА ЕЛЕКТРИЧНА СТРУКТУРНА



ДОДАТОК Б – СХЕМА ЕЛЕКТРИЧНА ПРИНЦИПОВА



ДОДАТОК В – СХЕМА ПРОГРАМИ



ДОДАТОК Г – КОД ПРОГРАМИ

// Підключення бібліотеки для роботи з датчиком BMP280

#include <Wire.h> // Бібліотека для роботи з шиною I2C

#include <SPI.h>  // Бібліотека для роботи з шиною SPI0

#include <Adafruit\_BMP280.h>

// Створення об'єкта для роботи з датчиком BMP280

Adafruit\_BMP280 bmp;

// Визначення пінів для підключення 7-сегментного LED

#define LED\_A 2

#define LED\_B 3

#define LED\_C 4

#define LED\_D 5

#define LED\_E 6

#define LED\_F 7

#define LED\_G 8

#define LED\_DP 9

#define LED\_D1 10

#define LED\_D2 11

#define LED\_D3 12

#define LED\_D4 13

// Визначення піну для підключення кнопки

#define BUTTON\_PIN 19

// Визначення масиву для зберігання значень тиску за останню годину

#define MAX\_SAMPLES 3600 // Кількість секунд у годині

float pressure\_samples[MAX\_SAMPLES];

int current\_sample = 0; // Індекс поточного значення тиску

// Визначення змінних для зберігання поточного, максимального та мінімального тиску

float current\_pressure = 0;

float max\_pressure = 0;

float min\_pressure = 0;

// Визначення змінної для зберігання поточного режиму відображення

int display\_mode = 0; // 0 - поточний тиск, 1 - максимальний тиск, 2 - мінімальний тиск

// Визначення змінної для зберігання стану кнопки

int button\_state = 0;

// Функція для ініціалізації датчика BMP280

void init\_bmp() {

  // Перевірка наявності датчика BMP280

  if (!bmp.begin()) {

    // Якщо датчика немає, то виводимо повідомлення про помилку "ЕЕЕ" і зупиняємо програму

     // Serial.println(“BMP280 SENSOR ERROR”);

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, LOW);

      digitalWrite(LED\_C, LOW);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, HIGH);

      digitalWrite(LED\_F, HIGH);

      digitalWrite(LED\_G, HIGH);

      digitalWrite(LED\_D1, LOW);

      digitalWrite(LED\_D2, LOW);

      digitalWrite(LED\_D3, LOW);

      while (1); //"Вічний" цикл (зупиняємо програму)

  }

}

// Функція ініціалізації 7-сегментного LED

void init\_led() {

  // Встановлення пінів для 7-сегментного LED як вихідних

  pinMode(LED\_A, OUTPUT);

  pinMode(LED\_B, OUTPUT);

  pinMode(LED\_C, OUTPUT);

  pinMode(LED\_D, OUTPUT);

  pinMode(LED\_E, OUTPUT);

  pinMode(LED\_F, OUTPUT);

  pinMode(LED\_G, OUTPUT);

  pinMode(LED\_DP, OUTPUT);

  pinMode(LED\_D1, OUTPUT);

  pinMode(LED\_D2, OUTPUT);

  pinMode(LED\_D3, OUTPUT);

  pinMode(LED\_D4, OUTPUT);

    // Вимкнення всіх сегментів та розрядів

  digitalWrite(LED\_A, LOW);

  digitalWrite(LED\_B, LOW);

  digitalWrite(LED\_C, LOW);

  digitalWrite(LED\_D, LOW);

  digitalWrite(LED\_E, LOW);

  digitalWrite(LED\_F, LOW);

  digitalWrite(LED\_G, LOW);

  digitalWrite(LED\_DP, LOW);

  digitalWrite(LED\_D1, HIGH);

  digitalWrite(LED\_D2, HIGH);

  digitalWrite(LED\_D3, HIGH);

  digitalWrite(LED\_D4, HIGH);

  }

// Функція для ініціалізації кнопки

void init\_button() {

  // Встановлення піну для кнопки як вхідного

  pinMode(BUTTON\_PIN, INPUT);

}

// Функція для читання значення тиску з датчика BMP280

void read\_pressure() {

  // Читання значення тиску з датчика BMP280 у гектопаскалях

  current\_pressure = bmp.readPressure() / 100.0;

  // Збереження поточного значення тиску у масиві

  pressure\_samples[current\_sample] = current\_pressure;

  // Збільшення індексу поточного значення тиску

  current\_sample++;

  // Якщо індекс досяг максимального значення, то обнулити його

  if (current\_sample == MAX\_SAMPLES) {

    current\_sample = 0;

  }

  // Пошук максимального та мінімального значення тиску у масиві

  //max\_pressure = pressure\_samples[0];

  //min\_pressure = pressure\_samples[0];

  // for (int i = 0; i < MAX\_SAMPLES; i++) {

  //   if (pressure\_samples[i] > max\_pressure) {

  //     max\_pressure = pressure\_samples[i];

  //    }

  //   if (pressure\_samples[i] < min\_pressure) {

  //    min\_pressure = pressure\_samples[i];

  //  }

  // }

}

// Функція для виводу значення тиску на 7-сегментний LED

void display\_pressure(float pressure) {

  // Конвертація значення тиску у ціле число

  int pressure\_int = (int)pressure;

  // Розбиття значення тиску на окремі цифри

  int digit1 = pressure\_int / 1000; // Тисячні

  int digit2 = (pressure\_int / 100) % 10; // Сотні

  int digit3 = (pressure\_int / 10) % 10; // Десятки

  int digit4 = pressure\_int % 10; // Одиниці

  // Виведення значення тиску на 7-сегментний LED

  display\_digit(digit1, LED\_D1); // Виведення тисячних

  display\_digit(digit2, LED\_D2); // Виведення сотень

  display\_digit(digit3, LED\_D3); // Виведення десятків

  display\_digit(digit4, LED\_D4); // Виведення одиниць

  digitalWrite(LED\_DP, HIGH); // Виведення десяткової крапки

}

// Функція для виводу цифри на 7-сегментний LED

void display\_digit(int digit, int pin) {

  // Включення потрібного розряду

  digitalWrite(pin, LOW);

  // Встановлення стану сегментів відповідно до цифри

  switch (digit) {

    case 0:

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, HIGH);

      digitalWrite(LED\_F, HIGH);

      digitalWrite(LED\_G, LOW);

      break;

    case 1:

      digitalWrite(LED\_A, LOW);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, LOW);

      digitalWrite(LED\_E, LOW);

      digitalWrite(LED\_F, LOW);

      digitalWrite(LED\_G, LOW);

      break;

    case 2:

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, LOW);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, HIGH);

      digitalWrite(LED\_F, LOW);

      digitalWrite(LED\_G, HIGH);

      break;

    case 3:

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, LOW);

      digitalWrite(LED\_F, LOW);

      digitalWrite(LED\_G, HIGH);

      break;

    case 4:

      digitalWrite(LED\_A, LOW);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, LOW);

      digitalWrite(LED\_E, LOW);

      digitalWrite(LED\_F, HIGH);

      digitalWrite(LED\_G, HIGH);

      break;

    case 5:

     digitalWrite(LED\_A, HIGH);

     digitalWrite(LED\_B, LOW);

     digitalWrite(LED\_C, HIGH);

     digitalWrite(LED\_D, HIGH);

     digitalWrite(LED\_E, LOW);

     digitalWrite(LED\_F, HIGH);

     digitalWrite(LED\_G, HIGH);

     break;

    case 6:

     digitalWrite(LED\_A, HIGH);

     digitalWrite(LED\_B, LOW);

     digitalWrite(LED\_C, HIGH);

     digitalWrite(LED\_D, HIGH);

     digitalWrite(LED\_E, HIGH);

     digitalWrite(LED\_F, HIGH);

     digitalWrite(LED\_G, HIGH);

     break;

    case 7:

     digitalWrite(LED\_A, HIGH);

     digitalWrite(LED\_B, HIGH);

     digitalWrite(LED\_C, HIGH);

     digitalWrite(LED\_D, LOW);

     digitalWrite(LED\_E, LOW);

     digitalWrite(LED\_F, LOW);

     digitalWrite(LED\_G, LOW);

     break;

    case 8:

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, HIGH);

      digitalWrite(LED\_F, HIGH);

      digitalWrite(LED\_G, HIGH);

      break;

    case 9:

      digitalWrite(LED\_A, HIGH);

      digitalWrite(LED\_B, HIGH);

      digitalWrite(LED\_C, HIGH);

      digitalWrite(LED\_D, HIGH);

      digitalWrite(LED\_E, LOW);

      digitalWrite(LED\_F, HIGH);

      digitalWrite(LED\_G, HIGH);

      break;

    default:

      digitalWrite(LED\_A, LOW);

      digitalWrite(LED\_B, LOW);

      digitalWrite(LED\_C, LOW);

      digitalWrite(LED\_D, LOW);

      digitalWrite(LED\_E, LOW);

      digitalWrite(LED\_F, LOW);

      digitalWrite(LED\_G, LOW);

      break;

  }

  // Вимкнення розряду

     digitalWrite(pin, HIGH);

}

// Функція для читання стану кнопки

void read\_button() {

  // Читання стану кнопки

  int new\_state = digitalRead(BUTTON\_PIN);

 // Якщо стан кнопки змінився, то змінити режим відображення

 if (new\_state != button\_state) {

  button\_state = new\_state;

    if (button\_state == HIGH) {

      display\_mode++;

        if (display\_mode > 2) {

          display\_mode = 0;

        }

    }

  }

}

// Функція для виконання основної логіки програми

void loop() {

  // Читання значення тиску з датчика BMP280

  read\_pressure();

  // // Читання стану кнопки

  read\_button();

  // Виведення значення тиску на 7-сегментний LED відповідно до режиму відображення

    switch (display\_mode) {

    case 0:

      display\_pressure(current\_pressure);

      break;

    case 1:

      //display\_pressure(max\_pressure);

      display\_pressure(1111);

      break;

    case 2:

      //display\_pressure(min\_pressure);

      display\_pressure(2222);

      break;

    default:

      break;

  }

}

// Функція для ініціалізації програми

void setup() {

  // Встановлення швидкості передачі даних через серійний порт

  Serial.begin(9600);

  // Ініціалізація 7-сегментного LED

  init\_led();

  // Ініціалізація датчика BMP280

  // init\_bmp();

  // Ініціалізація кнопки

  init\_button();

}