МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____(*Савченко А.С.*)
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"БАКАЛАВР"

Тема: Комп'ютерний пристрій контролю стану забруднення повітря

Виконавець: *Герус Роман Олександрович*

Керівник: **професор** *Зіатдінов Юрій Кашафович*

Нормоконтролер: **ст. викл.** *Шевченко О.П.*

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра Комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ (*Савченко А. С.)*
«____» _____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Геруса Романа Олександровича

1. Тема дипломного проекту: «Комп'ютерний пристрій контролю стану забруднення повітря» затверджена наказом ректора від «22» квітня 2021 р. № 636/ст.

2. Термін виконання роботи: з 10.05.2020 до 11.06.2020.

3. Вихідні дані до роботи: розробка комп'ютерного пристрою контролю стану забруднення повітря.

4. Зміст пояснювальної записки: вступ, аналіз сучасних підходів до контролю стану забруднення повітря, аналіз можливостей та засобів для створення комп’ютерного пристрою власноруч, створення макету комп’ютерного пристрою, висновок.

5. Перелік обов'язкового графічного матеріалу: налаштування середовища розробки, зображення обладнання, датчиків та плати.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1.	Аналіз літератури та джерел за темою дипломного проекту.	11.05.2020 14.05.2020	
2.	Розроблення та затвердження плану дипломного проекту.	15.05.2020 17.05.2020	
3.	Опрацювання інформації за тематикою.	18.05.2020 19.05.2020	
4.	Розробка розділу 1.	20.05.2020 23.05.2020	
5.	Розробка розділу 2.	24.05.2020 29.05.2020	
6.	Розробка розділу 3.	30.05.2020 04.06.2020	
7.	Написання пояснювальної записки. Підготовка графічного демонстраційного матеріалу.	05.06.2020 11.06.2020	

Студент **(*Герус Р.О.*)**

Керівник дипломного проекту **(*Зіатдінов Ю.К.)***

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Комп'ютерний пристрій контролю стану забруднення повітря»: 59 с.,26 рис., 8 літературних джерел.

Об’єкт дослідження: плата розробки Arduino Uno та датчики контролю стану повітря.

Предмет: розробка комп’ютерного пристрою контролю якості повітря, та його інтеграція в побутове життя.

Мета роботи: створення комп’ютерного пристрою для моніторингу якості повітря.

Методи дослідження: аналіз проблеми сучасного відношення до забруднення повітря.

Результат проекту – пристрій, створений для моніторингу та контролю якості повітря в житловому приміщенні.

ПОВІТРЯ, ARDUINO IDE, ARDUINO UNO, C++, UART, I2C, КОНТРОЛЬ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

EPA – управління з охорони навколишнього середовища США.

Arduino – компанія розробник плат розробки Arduino.

C ++ – сучасна низькорівнева мова програмування.

Simula – мова попередник C++.

Arduino IDE – інтегроване середовище розробки.

TX / RX – контакти, які відповідають за отримання та передачу даних протоколу UART.

SDA/SLC – контакти, які відповідають за отримання та передачу даних протоколу I2C.

NTC – терморезистор.

ВСТУП

Системи контролю якості повітря є у багатьох містах, у різних куточках світу. Але основною проблемою є те, що вони контролюють лише викиди великих об’ємів, не зважаючи при тому на проблему побутового забруднення повітря.

Стежити за викидами на побутовому рівні теж є важливою частиною нашого життя, адже наше здоров’я напряму залежить від того, чим ми дихаємо.

Метою дипломного проекту є створення комп’ютерного пристрою для моніторингу якості повітря.

Об’єктом дослідження є плата розробки Arduino Uno та датчики контролю стану повітря.

Для досягнення мети, яка стоїть перед нами, нам потрібно:

- Провести аналіз проблеми створення пристрою контролю стану забруднення повітря;
- зробити огляд недоліків та пропозицій щодо удосконалення підходів, методів та алгоритмів створення пристрою контролю стану забруднення повітря;
- проаналізувати існуючі методи контролю якості повітря;
- дослідити та обґрунтувати вибір технологій та інструментів для створення комп’ютерного пристрою контролю якості повітря;
- розробити прототип комп’ютерного пристрою.

Комп’ютерний пристрій повинен забезпечувати наступний функціонал:

- вимірювати показники забруднення повітря з кожного датчика;
- виводити результат на екран виводу;
- інформувати користувача про можливі помилки в роботі.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд проблеми забруднення повітря в наші дні.

У наш час системи контролю забруднення є скоріше необхідністю, ніж іграшкою для різного роду досліджень. Забруднювачі повітря генеруються як природними (наприклад, виверженням вулканів та пожежами), так і техногенними джерелами, включаючи мобільні та стаціонарні. Мобільні стосуються рухомих джерел, таких як літаки, поїзди та автомобілі, а стаціонарні - стаціонарних промислових джерел, таких як електростанції, заводи та інші об'єкти. Стаціонарні джерела додатково розбиваються на основні та місцеві джерела: основні джерела викидають або 10 або більше тон одного забруднювача повітря або 25 або більше тон комбінації забруднювачів повітря, тоді як місцеві джерела викидають менше 10 тон одного або менше забруднювачів більше 25 тон комбінації забруднюючих речовин. У досить високих концентраціях забруднюючі речовини, що утворюються як мобільними, так і стаціонарними джерелами, можуть спричинити негативні наслідки для атмосфери, навколишнього середовища та життя людини, такі як підвищення середньої глобальної температури, зменшення видимості атмосфери, зниження якості повітря та вплив на здоров’я людини. Агентство з охорони навколишнього середовища (EPA) запровадило Закон про чисте повітря (CAA), який накладав норми на обидва джерела забруднювачів повітря, щоб допомогти пом'якшити їх вплив на атмосферу, навколишнє середовище та життя людей.

Типи забруднюючих речовин, що регулюються EPA, поділяються на три групи, які включають критерії забруднювачів повітря, токсичних речовин у повітрі та парникових газів, а саме:

- критерії забруднювачів повітря - це група з шести загальних забруднювачів повітря - тобто тверді частинки (ПМ), фотохімічні окислювачі (наприклад, озон), оксид вуглецю, оксид сірки, оксид азоту та свинець - що може негативно вплинути на здоров'я та добробут населення, атмосферу та навколишнє середовище, а також навколишні споруди.
- Термін "токсичні речовини в повітрі" відноситься до переліку понад 180 забруднювачів повітря - наприклад, органічних хімічних речовин, летких органічних сполук (ЛОС), металів та металевих сполук, включаючи паливо, розчинники, ртуть, миш'як, азбест та бензол, що дає небезпечні наслідки для здоров'я та навколишнього середовища, навіть якщо вони присутні в незначних кількостях і виробляються меншою кількістю джерел порівняно з критеріями забруднюючих речовин.
- Під парниковими газами (ПГ) маються на увазі гази, включаючи діоксид вуглецю, хлорфторуглероди (ХФУ), метан та озон, які впливають на здоров’я людей та сприяють посиленню парникового ефекту на Землі та подальшому впливу на глобальний клімат.

Загалом, нормативні акти, що стосуються цих різних типів забруднювачів повітря, диктують деякі міркування, про які повинні пам’ятати галузі, щоб залишатись відповідними стандартам EPA - наприклад, впровадження необхідного та належного обладнання та систем контролю забруднення повітря.

1.2. Історія розвитку Arduino. Причина успіху та вклад у розвиток мікроконтролерів.

Arduino – одна з найбільш відомих та успішних у світі апаратна та програмна екосистема з відкритим кодом. Компанія пропонує цілий низку програмних засобів, апаратних платформ та документації, що дозволяє майже будь-кому відкрити для себе

світ embedded розробки та стати професіоналом своєї справи. Arduino – це популярний інструмент для розробки продуктів iot, а також один з найуспішніших інструментів для навчання у сфері stem. Сотні тисяч дизайнерів, інженерів, студентів, розробників та виробників по всьому світу використовують плати arduino для розробки у таких галузях як: музика, ігри, розумні іграшки, системи розумних домівок, сільське господарство, тощо.

Розробка Arduino спочатку була розпочата як дослідницький проект Массімо Банзі, Девідом Куартьєльєсом, томом ігое, джанлукою мартіно та девідом меллісом з інституту дизайну взаємодії в івреї на початку 2000-х років. Цей проект базується на мові processing, яка призначена для вивчення кодування в контексті візуального мистецтва, розробленого Кейсі Ріс та Беном Фраєм, а також дипломного проекту Ернандо Баррагана про електромонтажну дошку.

Перша плата Arduino була представлена в 2005 році, щоб допомогти студентам-розробникам, які не мали досвіду роботи в галузі електроніки та програмування мікроконтролерів, створити робочі прототипи для їх потреб. З тих пір він став найбільш популярним інструментом для розробки різного виду пристроїв, що використовується інженерами і навіть гігантами it індустрії.

Arduino – це перший широко розповсюджений апаратний проект із відкритим кодом, який був створений для створення спільноти. Саме завдяки цьому, спільнота допомагає поширити використання інструменту та скористатися внесками сотень людей, які допомагали налагоджувати код, писати приклади, створювати навчальні посібники, підтримувати інших користувачів на форумах та створювати тисячі груп по всьому світу. Компанія Arduino зазначає, що саме відкритість коду допомогла їй добитись таких колосальних результатів у розвитку їх апаратних та програмних продуктів.

З моменту заснування проекту Arduino було представлено багато нових плат, мікроконтролерів та бібліотек програмного забезпечення, що розширює спектр можливостей, доступних для спільноти. Сьогодні, більш ніж через десять років від заснування компанії, Arduino продовжує розробляти на оновлювати апаратне та програмне забезпечення з відкритим кодом.

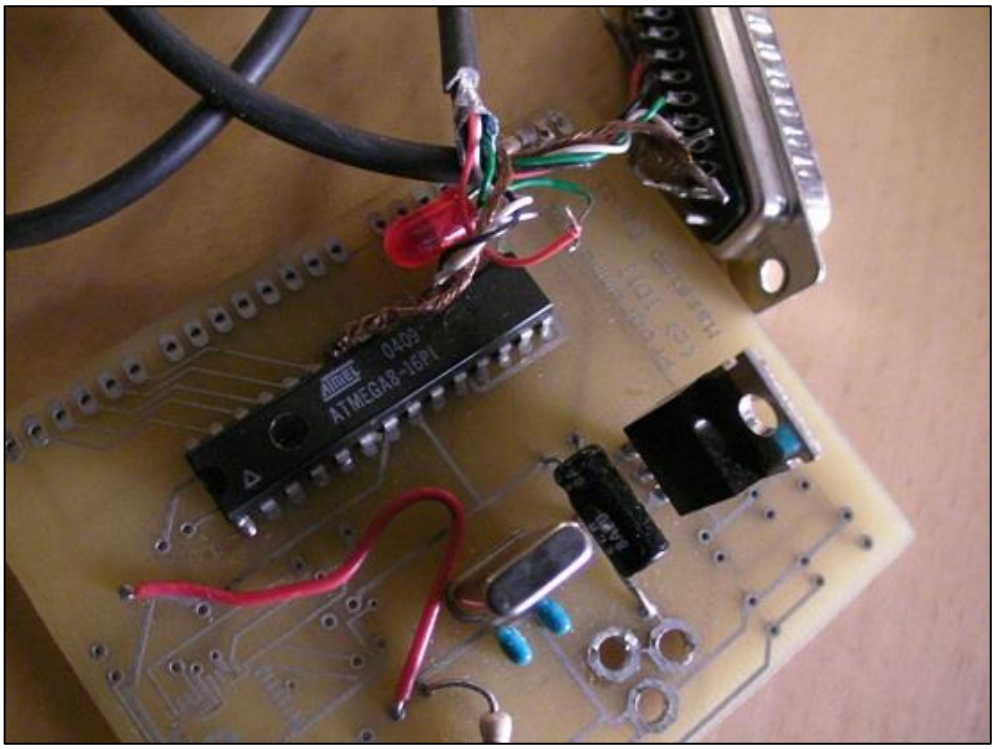

Рис. 1.1. Перша плата Arduino

1.3. Поняття мікроконтролеру, особливості його складових

Мікроконтролер - це компактна цілісна схема, призначена для управління конкретною операцією у вбудованій системі. Стандартний мікроконтролер включає в себе процесор, пам'ять та периферію вводу / виводу на одному чіпі.

Іноді їх називають вбудованим контролером або блоком мікроконтролера (mcu), мікроконтролери також часто зустрічаються в транспортних засобах, роботах, медичних пристроях, мобільних радіоприймачах, торгових автоматах та побутовій техніці. Одним слово, вони є простими, мініатюрними персональними комп’ютерами, призначеними для управління дрібними функціями більшого компонента, без складної операційної системи.

Мікроконтролер вбудований всередину системи для управління особливою функцією в пристрої. Він робить це, обробляючи дані, які отримує від периферійних пристроїв вводу-виводу, використовуючи центральний процесор. Тимчасова інформація, яку отримує мікроконтролер, зберігається в спеціальному сегменті пам’яті, який називається сегмент даних. Звідти процесор отримує до неї доступ і

використовує інструкції, що зберігаються в його програмній пам'яті, для розшифровки та застосування вхідних даних. Потім він використовує свої периферійні пристрої вводу-виводу для зв’язку та здійснення відповідних дій.

Мікроконтролери використовуються у великій кількості систем і пристроїв. У пристроях часто використовується кілька мікроконтролерів, які працюють разом для вирішення поставлених завдань.

Наприклад, в машині може бути багато мікроконтролерів, які управляють різними окремими системами, такими як антиблокувальна система гальмування автомобіля, система тяги, ін’єкція палива або контроль підвіски. Всі мікроконтролери взаємодіють між собою, щоб повідомити правильні дії. Деякі можуть спілкуватися з більш складним центральним комп'ютером у машині, а інші можуть спілкуватися лише з іншими мікроконтролерами. Вони надсилають та отримують дані за допомогою периферійних пристроїв вводу-виводу та обробляють ці дані для виконання призначених для них завдань.

Основними елементами мікроконтролера є:

- Процесор - процесор можна сприймати як мозок пристрою. Він обробляє та реагує на різні інструкції, що керують функціями мікроконтролера. Це передбачає виконання основних арифметичних, логічних операцій та операцій вводу-виводу. Він також виконує операції передачі даних.
- Пам'ять - пам'ять мікроконтролера використовується для зберігання даних, які отримує процесор, і використовує їх для того щоб дати відповідь на вказівки, які він запрограмував виконувати. Мікроконтролер має два основних типи пам'яті:
    1. Сегмент програмної пам’яті, яка зберігає довгострокову інформацію про інструкції, які виконує центральний процесор. Програмна пам’ять - це енергонезалежна пам'ять, тобто вона зберігає інформацію з часом, не потребуючи джерела живлення.
    2. Сегмент пам'яті даних, яка потрібна для тимчасового зберігання даних під час виконання інструкцій. Пам'ять даних є нестабільною, тобто дані,

які вона зберігає, є тимчасовими і зберігаються лише в тому випадку, якщо пристрій підключено до джерела живлення.

- Периферійні пристрої вводу-виводу - пристрої введення та виведення є інтерфейсом процесора до зовнішнього світу. Вхідні порти отримують інформацію та передають її процесору у вигляді двійкових даних. Процесор отримує ці дані і надсилає необхідні інструкції вихідним пристроям, які виконують зовнішні завдання для мікроконтролера.

Хоча процесор, пам’ять та периферія вводу-виводу є фундаментальними елементами мікропроцесора, є й інші елементи, які часто додаються до складу плат. Сам термін периферія вводу / виводу просто стосується допоміжних компонентів, які взаємодіють з пам'яттю та процесором. Є багато допоміжних компонентів, які можна класифікувати як периферійні пристрої.

Інші допоміжні елементи мікроконтролера включають:

- Аналого-цифровий перетворювач - ацп - це схема, яка перетворює аналогові сигнали в цифрові. Це дозволяє процесору в центрі мікроконтролера взаємодіяти із зовнішніми аналоговими пристроями, наприклад, такими як датчики;
- цифрово-аналоговий перетворювач - цап виконує зворотну функцію ацп і дозволяє процесору в центрі мікроконтролера передавати свої вихідні сигнали зовнішнім аналоговим компонентам;
- системна шина - це з'єднувальний провід, який з'єднує всі компоненти мікроконтролера разом;
- послідовний порт - послідовний порт є одним із прикладів порту вводу-виводу, що дозволяє мікроконтролеру підключатися до зовнішніх компонентів. Він має функцію, подібну до usb або паралельного порту, але відрізняється способом обміну бітами даних.

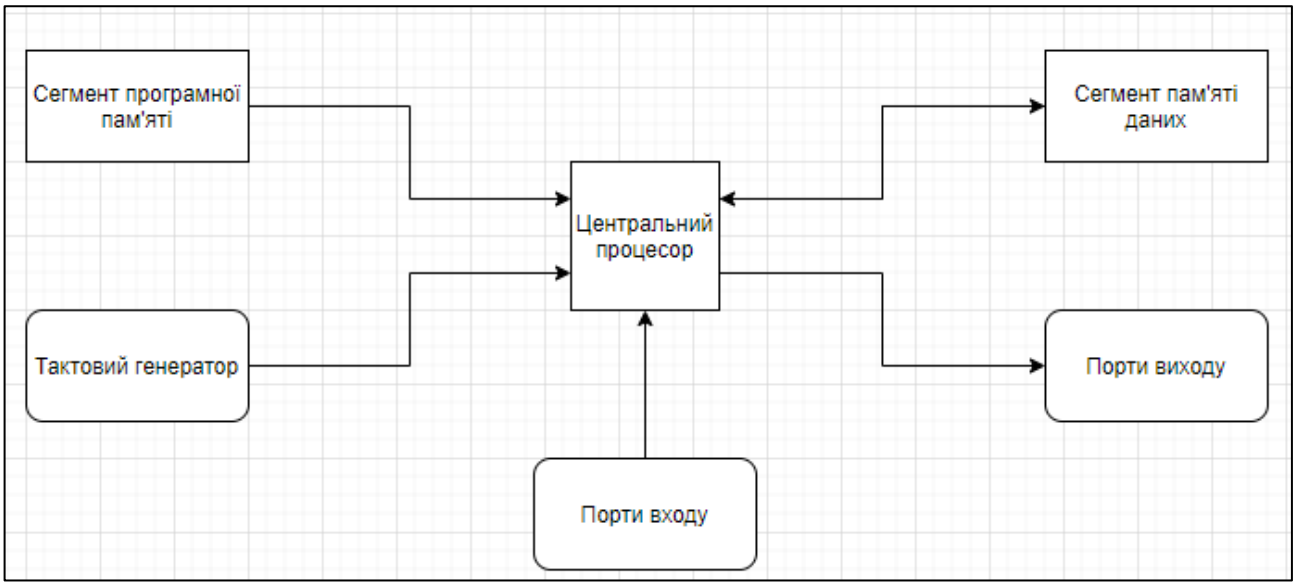

Рис. 1.2. Схема залежностей компонентів мікроконтролера

1.4. Класифікація мікроконтролерів.

Мікроконтролери характеризуються шириною шини, набором команд та структурою пам'яті. Для одного типу можуть існувати різні форми з різними джерелами.

Існують типи мікроконтролера, що характеризуються їх бітами, архітектурою пам'яті та набором команд.

1. Типи мікроконтролерів відповідно до кількості бітів

До цифрової ери вся електроніка була аналоговою. 8-ми бітний мікроконтролер мав справу з безперервним діапазоном напруги. У світі мікроконтролерів усі дані, пам’ять і програмний код є цифровими (0 і 1). Один біт - це 0 або 1. Колекція з восьми 0 та / або 1 відома як 8-біт (або байт).

16-розрядний мікроконтролер забезпечує більшу точність і продуктивність у порівнянні з 8-розрядним. Наприклад, 8-бітові мікроконтролери можуть використовувати лише 8 бітів, що призводить до кінцевого діапазону 0 × 00 - 0xff (0-255) для кожного циклу. На відміну від цього, 16-розрядні мікроконтролери з їх бітовою шириною даних мають діапазон 0 × 0000 - 0xffff (0-65535) для кожного циклу.

32-розрядний мікроконтролер використовує 32-розрядні інструкції для виконання арифметичних та логічних операцій. Вони використовуються в пристроях з автоматичним управлінням.

2. Типи мікроконтролерів відповідно до пристроїв пам'яті

Пристрої пам'яті поділяються на два типи:

- мікроконтролер вбудованої пам'яті;
- мікроконтролер зовнішньої пам'яті.

Мікроконтролер з вбудованою пам'яттю: коли вбудована система має блок мікроконтролера, який має всі функціональні блоки, доступні на мікросхемі, називається вбудованим мікроконтролером. Наприклад, 8-ми бітний мікроконтролер, що має сегмент програмної пам'яті та сегмент даних, порти вводу / виводу, послідовний зв’язок, лічильники та таймери та переривання на мікросхемі, є вбудованим мікроконтролером.

Мікроконтролер зовнішньої пам'яті: коли вбудована система має блок мікроконтролера, який має не всі функціональні блоки, доступні на мікросхемі, називається мікроконтролером зовнішньої пам'яті. Наприклад, деякі 8-ми бітні мікроконтролери не мають сегменту програмної пам'яті, тому він буде мікроконтролером зовнішньої пам'яті.

3. Типи мікроконтролерів відповідно до набору інструкцій

Cisc: cisc - це комп’ютер з складним набором інструкцій. Він дозволяє програмісту використовувати одну інструкцію замість багатьох простіших інструкцій для економії ресурсів та часу.

Risc: risc розшифровується як комп’ютер із зменшеною інструкцією. Цей тип наборів інструкцій зменшує конструкцію мікропроцесора для галузевих стандартів. Це дозволяє кожній інструкції працювати з будь-яким реєстром або використовувати будь-який режим адресації та одночасний доступ до програми та даних.

1.5. Поняття протоколу зв’язку, їх класифікація

Правильні описи форматів цифрових повідомлень, а також правила є відомими протоколами зв'язку. Основною функцією цих протоколів є обмін повідомленнями однієї комп’ютерної системи з іншою. Вони важливі для телекомунікаційних систем, оскільки вони послідовно надсилають та отримують повідомлення. Ці протоколи

забезпечують виявлення та виправлення помилок, сигналізацію та автентифікацію. Вони також можуть пояснити семантику, синтаксис та об'єднати аналогові та цифрові комунікації. Впровадження цих протоколів може здійснюватися як на апаратному, так і на програмному рівні. Виникає питання, чому протоколи зв’язку є настільки важливими? Завданням протоколів зв'язку – є допомога різноманітним мережевим пристроям комунікувати між собою, передаючи аналогові сигнали, цифрові сигнали, різні файли та обробляючи дані з одного пристрою на інші пристрої. Ці типи протоколів застосовуються в телекомунікаційних та комп’ютерних мережах. Наприклад, в глобальній мережі інтернет, найважливішими протоколами є tcp (протокол управління передачею) і протокол датаграм користувачів (udp).

Існує два типи протоколів зв'язку, які класифікуються нижче:

- міжсистемний протокол;
- внутрішньосистемний протокол.

Міжсистемний протокол використовується для зв'язку двох різних пристроїв. Наприклад, як спілкування між комп’ютером та набором мікроконтролера. Зв'язок здійснюється через міжшинову систему.

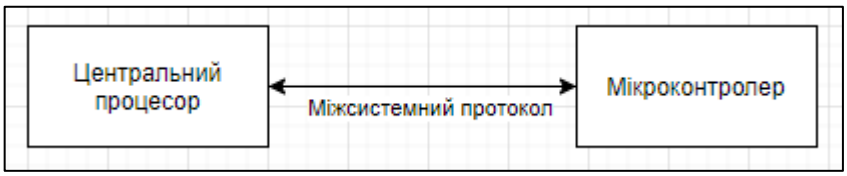

Рис. 1.3. Зв'язок через міжсистемний протокол.

Різні категорії міжсистемних протоколів в основному включають наступне.

- Протокол uart;
- протокол usart;
- протокол usb;

Uart - це універсальний асинхронний передавач і приймач. Протоколи uart - це послідовний зв’язок через два дротових протоколи. Сигнальні лінії кабелю даних позначаються як rx і tx. Послідовний зв’язок зазвичай використовується для передачі та

прийому сигналу. Він передається і отримує дані послідовно по бітах. Uart приймає байти даних і послідовно надсилає окремі біти.

Uart - це напівдуплексний протокол. Напівдуплекс означає передачу та отримання даних, але не одночасно. Більшість контролерів мають апаратно вбудований uart. Для передачі та прийому даних використовується лише одна лінія даних. Цей протокол має один стартовий біт, 8-бітові даних та стоп біт.

Usart - це універсальний синхронний та асинхронний передавач та приймач. Це послідовний зв’язок двопровідного протоколу. Сигнальні лінії кабелю даних позначаються як rx і tx. Цей протокол використовується для передачі та прийому даних байт за байтом разом із тактовими імпульсами. Це повнодуплексний протокол, який означає передачу та отримання даних одночасно з різною швидкістю.

Usb - це універсальна послідовна шина. Знову ж таки, це послідовний зв'язок двопровідного протоколу. Сигнальні лінії кабелю даних позначаються d + і d-. Цей протокол використовується для зв'язку з периферійними пристроями системи. Протокол usb використовується для послідовного надсилання та отримання даних на хост і периферійні пристрої. Для зв'язку usb потрібне програмне забезпечення драйверів. Пристрої usb можуть передавати дані на шину без будь-якого запиту на хост-комп'ютері.

Зараз більшість пристроїв щодня використовують usb протокол. Usb передає дані в різних режимах. Перший - це режим повільної швидкості від 10 кбіт / с до 100 кбіт /с; другий - повношвидкісний режим від 500 до 10 мбіт / с, швидкісний режим від 25 до 400 мбіт / с. Максимальна довжина кабелю usb - 4 метри.

Внутрішньосистемний протокол використовується для зв'язку двох пристроїв на готовій, друкованій платі. Використовуючи ці внутрішньосистемні протоколи, не переходячи до міжсистемних протоколів, ми розширяємо периферію мікроконтролера. Складність схеми та споживання енергії будуть збільшені за допомогою внутрішньосистемного протоколу. Використовуючи внутрішньосистемні протоколи, складність ланцюга та споживання електроенергії, знижують вартість, і це дуже безпечно для доступу до даних.

Різні категорії внутрішньосистемних протоколів в основному включають наступні протоколи:

- Протокол i2c;
- протокол spi;
- протокол can.

I2c розшифровується як інтегральна схема, і йому потрібні лише два проводи, які підключають всю периферію до мікроконтролера. I2c вимагає двох проводів sda (послідовна лінія передачі даних) і scl (послідовна лінія тактування) для передачі інформації між пристроями. Протокол працює за принципом «from master to slave», що розшифровується як «від головного пристрою до підлеглих». Головний пристрій надсилає адресу цільового веденого пристрою і зчитує / записує мітку. Адреса відповідає будь-якому підлеглому пристрою, коли пристрій увімкнено, решта підлеглих пристроїв в цей час - вимкнені .

Після того, як адреса збігається, зв’язок між головним пристроєм та підлеглим пристроєм продовжується, а також передаються та приймаються дані. Передавач передає 8-бітові дані, приймач відповідає 1-бітовим підтвердженням. Після завершення зв'язку головний пристрій видає умову зупинки. Шина i2c була розроблена компанією philips semiconductors. Її початкова мета - забезпечити простий спосіб підключення процесора до чіпів периферії.

Spi - це послідовний периферійний інтерфейс. Це один із багатьох послідовних протоколів зв'язку, розроблених компанією motorola. Іноді протокол spi також називають 4-провідним протоколом. Для цього потрібні чотири дроти mosi, miso, ss та протокол sclk.spi, що використовуються для зв'язку головного та підпорядкованого пристроїв. Головний пристрій спочатку послідовну лінію тактування з використанням частоти. Потім головний пристрій вибирає конкретний підпорядкований йому пристрій для зв'язку. Одночасно, головний пристрій може мати підключеного лише одного підпорядкованого йому пристрою. Це повнодуплексний протокол зв'язку. Він не обмежується 8-бітовими словами у разі передачі бітів.

Can – це протокол послідовного зв'язку. Для успішної передачі даних потрібні два дроти can high (h +) і can low (h-). Цей протокол був розроблений компанією robert bosh в 1985 році для автомобільних мереж.

1.6. Поняття мультиплексора. Принцип роботи.

Мультиплексування - загальний термін, що використовується для опису операції надсилання одного або декількох аналогових або цифрових сигналів по загальній лінії передачі в різний час або швидкість, і пристрій, який ми використовуємо для цього, називається мультиплексор.

Мультиплексор, скорочений до “MUX” або “MPX”, являє собою комбінаційну логічну схему, призначену для перемикання однієї з декількох вхідних ліній в єдину загальну вихідну лінію шляхом застосування керуючого сигналу. Мультиплексори працюють як дуже швидкодіючі багатопозиційні поворотні перемикачі, що з'єднують або керують декількома вхідними лініями, які називаються "каналами" по одному на вихід. Мультиплексори, або MUX, можуть бути або цифровими схемами, виготовленими з високошвидкісних логічних входів, що використовуються для перемикання цифрових або двійкових даних, або вони можуть бути аналоговими типами за допомогою транзисторів для перемикання одного з входів напруги або струму на один вихід.

Найбільш базовим типом мультиплексора є односторонній поворотний перемикач, як показано на рис. 1.4.

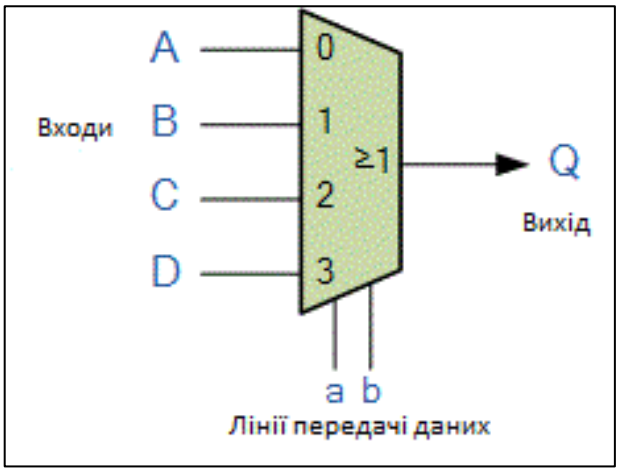

Рис. 1.4. Односторонній поворотний перемикач (мультиплексор)

Поворотний перемикач, який також називають вафельним перемикачем, оскільки кожен шар перемикача відомий як напівпровідникова пластина, є механічним пристроєм. Іншими словами, поворотний перемикач - це ручний перемикач, який ви можете використовувати для вибору окремих даних або сигнальних ліній, просто ввімкнувши його входи «УВІМК.» Або «ВИМК.». Тож як ми можемо автоматично вибрати кожен вхід даних за допомогою цифрового пристрою.

У цифровій електроніці мультиплексори також відомі як селектори даних, оскільки вони можуть “вибирати” кожен вхідний рядок, будуються з окремих аналогових комутаторів, укладених в єдиний пакет мікросхеми, на відміну від селекторів “механічного” типу, таких як звичайні перемикачі.

Вони використовуються як один із методів зменшення кількості логічних затворів, необхідних у схемі, або коли для передачі двох або більше різних цифрових сигналів потрібна одна лінія даних або шина даних. Наприклад, один 8-канальний мультиплексор.

Як правило, вибір кожного вхідного рядка в мультиплексорі контролюється додатковим набором входів, які називаються лініями управління, і відповідно до бінарних умов цих входів управління, “HIGH” або “LOW”, відповідний вхід даних підключається безпосередньо до виходу. Зазвичай мультиплексор має парну кількість 2n рядків введення даних та кількість входів "управління", які відповідають кількості входів даних.

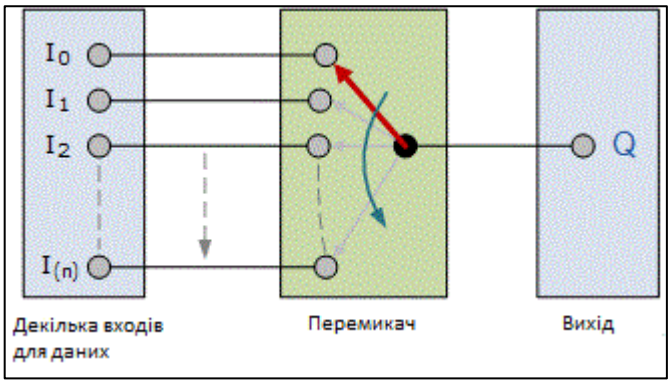

Рис. 1.5. Схема перемикача в мультиплексорі

Для кожного виду мультиплексора існує таблиця істинності. Наприклад, таблиця істинності мультиплексора «2 до 1» наведена нижче. Залежно від значення вибраного входу, на входах створюються входи, тобто D0, D1. Вихід - D0, коли значення вибору - S = 0, а результат - D1, коли значення вибору - S = 1.

Таблиця 1.1

Таблиця істинності для мультиплексора виду «2 до 1»

S	D0	D1	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

«X» у наведеній вище таблиці істинності позначає стан, який не є важливим. Отже, ігноруючи X, ми можемо отримати булевий вираз типового мультиплексора «2 до 1» таким чином: Y = SD0 + SD1

1.7. Висновки до розділу.

Отже, у наш час контроль якості забруднення повітря відіграє важливу роль у нашому повсякденному житті. Зараз існує багато світових організацій, які виводять

тему екології на державний та всесвітній рівень. Внаслідок різкого збільшення викидів парникових газів, органічних хімічних речовин та різного виду пилу, створилось багато центрів, які контролюють якість повітря. Але проблемою є те, що ці центри контролюють якість повітря лише біля великих заводів, місць видобування корисних копалин і т.д. Наразі не є поширеною тема розробки пристроїв, які б контролювали кількість небезпечних та шкідливих речовин, які є, наприклад, в нашому будинку, університетській аудиторії чи офісі.

Для розробки такого виду пристрою - необхідно розуміти основні принципи роботи мікроконтролерів та їх види. Також, при виборі датчиків для моніторингу якості повітря необхідно враховувати їх сумісність з обраним вами мікроконтролером, а саме:

- протоколи, за допомогою яких вони збирають та надсилають дані;
- їх термін роботи;
- правильно відкалібрувати датчики, для отримання коректних результатів.

Також, нам необхідно враховувати середовище, де ми будемо використовувати наш пристрій, чи буде це кімната в будинку, чи це пристрій для зовнішнього використання. Адже буде кардинально відрізнятись вибір датчиків та процес розробки та використання такого пристрою.

РОЗДІЛ 2
ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОГО ПРИСТРОЮ КОНТРОЛЮ СТАНУ ЗАБРУДНЕННЯ ПОВІТРЯ.

2.1. Arduino uno. Переваги та недостатки.

На початку 2000-х років, якщо ви хотіли розробити робота-іграшку, вам потрібно було обрати лінійку мікроконтролерів BASIC Stamp, вартість якої близько 100 доларів. Це може бути великими витратами, наприклад, для студентів, які завжди цікавляться новинками в технологічному світі та йдуть в ногу з часом. П'ятнадцять років потому, плати Arduino допомогли підживити рух Maker завдяки платформам для розробки, які можна придбати за ціною від 25 доларів США або навіть менше. Здається, Arduino Uno є певним стандартом для розробників початківців, і будь хто, охочий почати експериментувати з мікроконтролерами та робототехнікою повинен обирати саме Uno.

2.1.1. Технічні характеристики.

Arduino Uno використовує для обробки мікроконтролер ATmega328P і може живитися від 6 до 20 вольт (постійного струму), хоча рекомендується залишатися між 7 і 12. Він має 14 роз'ємів вводу-виводу, 6 з яких можна використовувати для широтно-імпульсної модуляції (ШІМ). Крім того, він має 6 аналогових вхідних роз'ємів.

Він програмується за допомогою IDE Arduino - доступний у Windows, MacOS, Linux або через веб-програму - за допомогою стандартного кабелю USB. Якщо ви маєте одну з цих плат і хочете перейти до її програмування, пристрій має вбудовану лампочку на роз'ємі 13. Ви можете просто запрограмувати індикатор блимати. Це чудовий приклад першої програми або може служити як корисний діагностичний інструмент, для того, щоб перевірити роботоздатність плати.

2.1.2. Переваги ARDUINO UNO над іншими платами для розробки.

На мою думку, найбільшим плюсом є те, що почати вивчення світу Embedded розробки з цієї плати дуже легко, і існує величезна кількість ресурсів, які допоможуть вам у цьому. Насправді, пошуковий запит у Google вигляду "початок роботи з Arduino" дає 553 000 результатів на момент написання цієї роботи.

Інша велика перевага для досвідчених користувачів - кількість доступних екранів - плат розширення, що підключаються до плати розробки. Наприклад, якщо ви хочете керувати різними типами двигунів, для цього є готова плата розширення. Також, плати Arduino порівняно недорогі в порівнянні з іншими платформами мікроконтролерів. Найменш дорога версія модуля Arduino може бути зібрана вручну, і навіть попередньо зібрані модулі Arduino коштують менше 50 доларів.

Крос-платформеність також є великим плюсом плат від Arduino. Програмне забезпечення Arduino працює на операційних системах Windows, Macintosh OSX та Linux. Більшість систем інших мікроконтролерів обмежені Windows. Середовище програмування Arduino є простим у використанні для початківців, але при цьому досить гнучким для досвідчених користувачів. Студенти, які навчаються програмуванню в цьому середовищі, будуть добре знайомі із зовнішнім виглядом Arduino та її функціями.

Великою перевагою є те, що бібліотеки Arduino є open source, тобто бібліотеки з відкритим кодом, та вони доступні для розширення досвідченими програмістами. Мову можна розширити за допомогою бібліотек C ++, і люди, які хочуть зрозуміти технічні деталі, можуть зробити перехід від Arduino до мови програмування AVR C,

на якій вона базується. Подібним чином, ви можете додати код AVR-C безпосередньо у свої програми Arduino. Arduino заснований на мікроконтролерах ATMEGA8 та ATMEGA168 від Atmel. Плани модулів публікуються під ліцензією Creative Commons, тому досвідчені дизайнери схем можуть зробити власну версію модуля, розширивши його та вдосконаливши. Навіть відносно недосвідчені користувачі можуть створити макетну версію модуля, щоб зрозуміти, як він працює, і заощадити свої кошти на покупці готового.

2.1.3. Недостатки ARDUINO UNO в порівнянні з іншими платами для розробки.

З іншого боку, порівняно з іншими платами розробки, тактова частота 16 МГц не є вражаючою. Це, ймовірно, не буде проблемою для робота, який, скажімо, поливає рослину кожні два дні, але в деяких більш залучених схемах управління це може представляти певні труднощі. З тієї ж точки зору, пам’ять на 32 КБ - це не багато місця, але знову ж таки, це залежить від вашої програми.

Крім того, немає вбудованого з’єднання Wi-Fi або Bluetooth, але їх можна додати за потреби. Хоча деякі аспекти можуть здаватися застарілими, багато в чому саме Uno започаткував ідею, що люди можуть робити дивовижні речі за розумну ціну. Ви можете придбати одну з цих плат за ціною близько 5 доларів США, або, придбати клон, яких дуже багато.

2.2. Мова програмування C++. Загальний огляд, причини вибору для реалізації завдання.

Мова програмування C ++ надає модель пам'яті та обчислень, яка точно відповідає моделі більшості комп'ютерів. Крім того, вона забезпечує потужні та гнучкі механізми абстракції; тобто конструкції, що дозволяють програмісту вводити та використовувати нові типи об’єктів, що відповідають концепціям програми. Таким чином, C ++ підтримує стилі програмування, які покладаються на досить пряме

маніпулювання апаратними ресурсами, щоб забезпечити високий ступінь ефективності, а також стилі програмування вищого рівня, що стосуються лише визначених користувачем типів, щоб надати модель даних та обчислень, що ближче до погляду людини на завдання, яке виконує комп'ютер. Ці стилі програмування вищого рівня часто називають абстракцією даних, об’єктно-орієнтованим програмуванням та загальним програмуванням.

2.2.1. Історія виникнення та розвиток мови програмування С++.

С ++ був розроблений і впроваджений Б’ярном Страуструпом в лабораторіях AT&T Bell, щоб поєднати організаційні та конструктивні переваги Simula з можливостями мови програмування С для програмування систем. Початкова версія С ++, яка називається «С з класами», вперше була використана в 1980 році; вона підтримувала традиційні методи системного програмування та абстракції даних. Основні зручності для об'єктно-орієнтованого програмування були додані в 1983 р., а об'єктно-орієнтовані методи проектування та програмування впроваджувались в спільноту С ++ поступово. Мова стала вперше комерційно доступна у 1985 році. Засоби загального програмування були додані до мови в період 1987-1989 рр..

В результаті широкого використання та появи декількох незалежно розроблених реалізацій С ++, офіційна стандартизація С ++ була розпочата в 1990 році під егідою Американського інституту національних стандартів ANSI, а пізніше Міжнародної організації зі стандартів ISO, що призвело до виникнення першого міжнародного стандарту в 1998 році – С++98. Протягом періоду стандартизації комітет зі стандартів виступав важливим акцентом для спільноти С ++, а його проекти стандартів діяли як проміжні визначення мови.

С ++ був розроблений, щоб забезпечити гнучкість та ефективність С для програмування систем разом із засобами Simula для організації програм (зазвичай їх називають об'єктно-орієнтованим програмуванням). З великою обережністю було застосовано те, що технології програмування вищого рівня від Simula можуть бути застосовані до домену системного програмування. Тобто механізми абстрагування,

надані C ++, були спеціально розроблені, щоб бути застосовними до завдань програмування, що вимагали найвищого ступеня ефективності та гнучкості.

C ++ був розроблений, щоб забезпечити гнучкість та ефективність у розробці різного виду систем. З великою обережністю було застосовано те, що технології програмування вищого рівня від Simula можуть бути застосовані до домену системного програмування. Тобто механізми абстракції, надані C ++, були спеціально розроблені, щоб бути застосовними до завдань програмування, що вимагали найвищого ступеня ефективності та гнучкості. Підтримка загального програмування з'явилася пізніше.

Метою C ++ було покращення якості програм, що створюються шляхом спрощення методів проектування та програмування. Більшість із цих методів походять з Simula і зазвичай обговорюються під мітками об'єктно-орієнтованого програмування та об'єктно-орієнтованого проектування. Однак метою завжди було підтримувати цілий ряд стилів дизайну та програмування. Це контрастує з поглядом на мовний дизайн, який намагається спрямувати всю побудову системи в єдиний сильно підтримуваний і дотриманий стиль.

Розробник цієї мови програмування описував загальні правила, яких потрібно дотримуватись, коли розробляєш програмне забезпечення для певної системи, а саме:

- Відсутність неявних порушень системи статичного типу;
- при розробці надавати таку ж підтримку для визначених користувачем типів, як і для вбудованих типів;
- використання локальних змінних;
- якщо ви сумніваєтеся, виберіть варіант функції, яку найпростіше втілити;
- синтаксис має значення;
- необхідно виключити використання препроцесора.

Ці правила слід розглядати в контексті, створеному для більш загальних цілей. Зокрема, бажання забезпечити високий ступінь сумісності C, безкомпромісну ефективність та негайну реальну утиліту протидіє бажанням забезпечити повну безпеку типу, повну загальність та абстрактну красу. З Simula C ++ запозичив поняття визначених користувачем типів та ієрархії класів. Однак у Simula та багатьох подібних мовах існують принципові відмінності у підтримці, що надається для визначених

користувачем типів та для вбудованих типів. Наприклад, Simula не дозволяє розміщувати в стеку об’єкти визначених користувачем типів і безпосередньо звертатися до них. Натомість усі об’єкти класу мають бути розподілені в динамічній пам’яті та доступні за допомогою покажчиків (що називаються посиланнями в Simula). І навпаки, вбудовані типи можуть бути справді локальними, не можуть бути розподілені в динамічній пам’яті і на них не можуть посилатися вказівники. Ця різниця у вбудованих типів та визначених користувачем типів мала наслідки серйозної ефективності. Наприклад, коли тип представляють як посилання на об’єкт, виділений у динамічній пам’яті, визначений користувачем тип - спричиняє накладні витрати під час виконання, які були визнані неприйнятними для виду програм, для яких призначений C ++. Крім того, різниця в стилі використання виключатиме однакову обробку семантично подібних типів у загальному програмуванні. Отже, C ++ надає класи, простори імен та контроль доступу, щоб допомогти локалізувати дизайнерські рішення. Деякі залежності неминучі в мові, призначеній для однопрохідної компіляції. Наприклад, у C ++ змінну або функцію не можна використовувати до того, як їх буде оголошено. Однак правила щодо імен членів класів та правила вирішення перевантажень були зроблені незалежними від порядку декларації, щоб мінімізувати плутанину та помилки.

2.2.2. Управління пам’яттю в C++. Переваги над іншими мовами програмування.

Вибір багатьох розробників припадає саме на C++, бо ця мова програмування має ручне управління пам’яттю. Це дозволяє програмісту власноруч контролювати ресурси платформи, під яку розробляється програмний продукт. Існує два способи виділення пам’яті для зберігання даних.

- Розподіл часу компіляції або статичний розподіл пам’яті: де пам’ять для названих змінних виділяється компілятором. Точний розмір і пам'ять повинні бути відомі під час компіляції, а для оголошення масиву розмір повинен бути постійним;

- виділення під час виконання або динамічний розподіл пам'яті: де пам'ять виділяється під час виконання, а виділення місця в пам'яті здійснюється динамічно в межах запуску програми, а сегмент пам'яті називається купою або вільним сховищем. У цьому випадку точний простір або номер елемента не повинен знати компілятор заздалегідь. Покажчики відіграють головну роль у цій справі.

У С ++ пам'ять, яка використовується додатком, розділена на п'ять контейнерів:

- текстовий сегмент - містить набір інструкцій для виконання програми.
- Сегмент даних - містить статичні та глобальні змінні, ініційовані значеннями, наприклад float pi = 3,14; - розмір сегмента даних заздалегідь визначається під час компіляції програми і залежить від розміру змінних у вихідному коді.
- bss (блок, розпочатий символом) сегмент - містить статичні та глобальні змінні, не явно ініціалізовані будь-яким значенням, наприклад int n; - розмір сегмента даних заздалегідь визначається під час компіляції програми і залежить від розміру змінних у вихідному коді.
- Стек - структура LIFO (останній зайшов, перший вийшов), що містить виклики функції та змінні, ініціалізовані у функціях - розмір стека визначається заздалегідь під час компіляції програми і залежить від операційної системи та середовища розробки.
- Heap (купа) - містить всю динамічно виділену пам'ять (наприклад, за допомогою інструкції new у С ++).

Доступ до даних у купі підтримується за допомогою покажчиків, які зберігають адресу пам'яті цих даних. Якщо вказівник перестає існувати (наприклад, через те, що функція, що його містить, виконалась, а тому була вилучена зі стеку), програміст не має можливості знову посилатися на дані в купі. Невикористані, недоступні дані залишатимуться в пам'яті, споживаючи цінні ресурси, якщо їх неправильно вивільнити за допомогою інструкції видалення. Проблеми можуть виникнути, якщо одна і та ж адреса пам'яті утримується двома різними вказівниками - вивільнення пам'яті може зробити один із вказівників порожнім і, можливо, призвести до помилок.

З метою полегшення управління пам’яттю стандарт C ++ 11 запровадив спільні вказівники (shared_ptr). Спільні вказівники підтримують лічильник посилань, який збільшується, коли інший спільний вказівник посилається на ту ж адресу пам'яті. Звільнення пам’яті буде здійснено лише тоді, коли всі спільні вказівники, що посилаються на пам’ять, будуть знищені.

2.3. Середовище розробки ARDUINO IDE. Основні функції.

2.3.1. Загальний огляд середовища розробки. Приклад тестової програми.

Arduino IDE (інтегроване середовище розробки) використовується для написання комп'ютерного коду та завантаження цього коду на фізичну плату. Arduino IDE дуже проста, і ця простота є, мабуть, однією з головних причин, чому Arduino став настільки популярним. Ми, безумовно, можемо стверджувати, що сумісність з Arduino IDE зараз є однією з головних вимог до нової плати мікроконтролера. Протягом багатьох років до Arduino IDE було додано багато корисних функцій, і тепер ви можете керувати сторонніми бібліотеками та платами з IDE, і при цьому зберігати простоту програмування плати. Головне вікно IDE Arduino показано нижче.

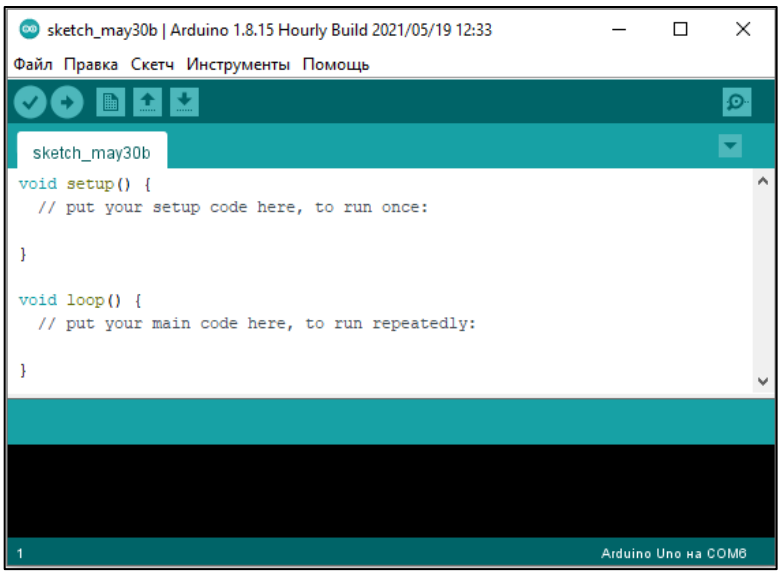

Рис. 2.1. Головне вікно Arduino IDE

Проекти, зроблені з використанням Arduino, називаються ескізами, і такі ескізи, як правило, пишуться у скороченій версії C ++ (ряд функцій C ++ не включаються). Оскільки програмування мікроконтролера дещо відрізняється від програмування комп'ютера, існує ціла низка бібліотек, специфічних для пристрою (наприклад, зміна режимів виведення, вихідні дані на висновках, зчитування аналогових значень та таймери). Це іноді бентежить користувачів, які вважають, що Arduino запрограмований на "мові Arduino". Однак Arduino насправді запрограмований на C ++. Він просто використовує унікальні бібліотеки для пристрою.

Як видно з рисунку 2.2, на панелі є шість кнопок, які відповідають за основні функції роботи з платою. Хоча більш просунуті проекти будуть використовувати переваги вбудованих інструментів в IDE, більшість проектів будуть спиратися саме на ці шість кнопок, розташованих під рядком меню. Розглянемо доступні кнопки:

Рис. 2.2. Кнопки, які відповідають за основні функції

- Прапорець використовується для підтвердження вашого коду. Клацніть на нього, як тільки ви напишете свій код;
- стрілка завантажує ваш код на Arduino для запуску;
- іконка з листом паперу створить новий файл;
- стрілка вгору використовується для відкриття існуючого проекту Arduino;
- стрілка вниз використовується для збереження поточного файлу;
- крайня права кнопка - це послідовний монітор, який корисний для надсилання даних з Arduino на ПК.

Існує безліч інших функцій, які можна розглянути в IDE. Але, використовуючи безліч різних типів мікроконтролерів та беручи участь у багатьох середовищах програмування, це шокує, наскільки простими є Arduino та його IDE! Менш ніж за дві

хвилини ви можете завантажити просту програму, написану за допомогою мови програмування C ++, на Arduino і запустити її.

2.3.2. Приклад тестової програми.

Розглянемо тестову програму, після встановлення якої, на платі розробки буде блимати світлодіод з інтервалом у 1 секунду. Проініціалізуємо функцію setup(). Функція setup() викликається при запуску програми. Використовується для ініціалізації змінних, початку використання бібліотек тощо. Функція setup() запускатиметься лише один раз після кожного включення або скидання плати Arduino.

```
// функція setup запускається один раз, коли ви натискаєте кнопку скидання або живлення плати
void setup() {
  // ініціалізувати цифровий контакт LED_BUILTIN як вихід.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Рис. 2.3. Ініціалізація функції setup()

Наступним кроком стане ініціалізація функції loop(). Після створення функції setup(), яка ініціалізує та встановлює початкові значення, функція loop() виконує саме те, що підказує її назва, і виконує цикли послідовно, дозволяючи вашій програмі змінюватись та реагувати. Використовується для активного управління платою Arduino.

```
// функція loop працює знову і знову
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // увімкається світлодіод (HIGH - рівень напруги)
  delay(1000);                       // чекаємо секунду
  digitalWrite(LED_BUILTIN, LOW);    // вимикається світлодіод, зробивши напругу низькою
  delay(1000);                       // чекаємо секунду
}
```

Рис. 2.4. Ініціалізація функції loop()

2.4. Вибір датчиків для контролю якості повітря. Їх характеристики.

2.4.1. Датчик контролю вмісту пилу у повітрі.

Для вимірювання вмісту пилу у повітрі я обрав датчик фірми PlanTower, моделі PMS1003. PMS1003 - це датчик якості повітря. Він виявляє значення концентрації PM 1,0, PM 2,5, PM 10. Цей пристрій може виявити концентрацію частинок від 0,3 до 10 мікрометрів у повітрі. Датчик виводить свої дані через UART.

Візьмемо, наприклад, PM 2,5 - це дрібні частинки діаметром 2,5 мікрометра або менше. У PMS1003 є 8 контактів:

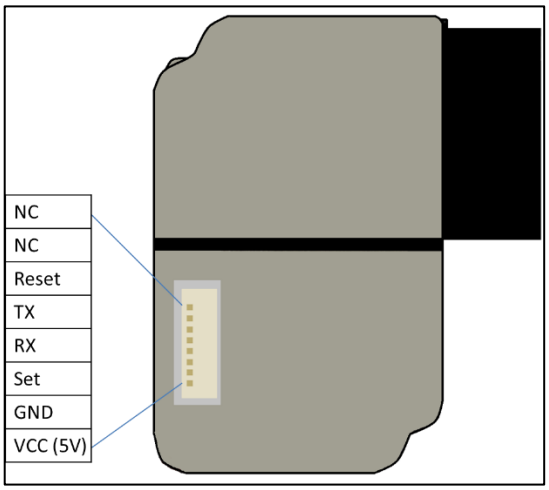

Рис. 2.5. Контакти датчика PMS1003

PMS1003 вимагає живлення 5 В, але робоча напруга його логічних входів становить 3,3 В. Отже, робоча напруга Reset, TX, RX, Set також становить 3,3 В.

Якщо контакт “Set” підтягнути, PMS1003 переводиться в робочий режим. Якщо контакт “Set” опустити, PMS3003 переходить у режим очікування.

Виводи TX / RX призначені для підключення UART. У робочому режимі PMS1003 видає дані, які він постійно читає. Розмір даних складає 32 байти.

Рис. 2.6. Зовнішній вигляд датчика PMS1003

2.4.2. Датчик контролю вологості та температури.

Для вимірювання температури та вологості повітря я обрав датчик фірми DHT11. DHT11 - це базовий, над дешевий цифровий датчик температури та вологості. Він використовує ємкісний датчик вологості та терморезистор для вимірювання навколишнього повітря і видає сигнал на контакт даних. Він досить простий у використанні, але вимагає ретельного часу для збору даних. Єдиним реальним мінусом цього датчика є те, що ви можете отримувати нові дані з нього лише раз на 2 секунди, показання датчика можуть бути до 2 секунд. Технічні деталі датчика:

- низька вартість;
- потужність від 3 до 5 В;
- точний для показань вологості 20-80% з точністю 5%;
- точний для показань температури 0-50 ° C з точністю ± 2 ° C;
- частота дискретизації не більше 1 Гц (раз на секунду);
- розмір корпусу 15,5 мм x 12 мм x 5,5 мм.

Як DHT11 вимірює вологість та температуру? DHT11 виявляє водяну пару, вимірюючи електричний опір між двома електродами. Компонент, що сприймає вологість, - це корпус, що утримує вологу, з нанесеними на поверхню електродами. Коли водяна пара поглинається підкладкою, іони вивільняються підкладкою, що збільшує провідність між електродами. Зміна опору між двома електродами пропорційна відносній вологості. Більш висока відносна вологість зменшує опір між електродами, тоді як менша відносна вологість збільшує опір між електродами.

DHT11 вимірює температуру за допомогою вбудованого в пристрій датчика температури NTC (терморезистора).

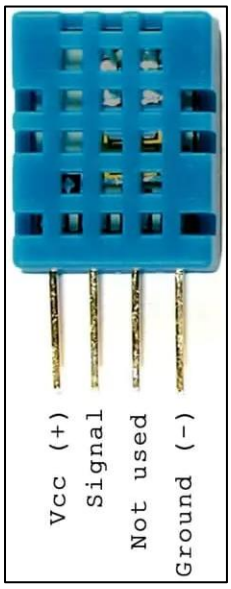

Рис 2.7. Контакти датчика DHT11

- VCC - підключається до мережі 3,3 - 5 В. Деколи потужності 3,3 В недостатньо, і в цьому випадку обирається потужність 5 В;
- data out – дані, які ми отримуємо на виході;
- третій контакт не використовується;
- ground – відповідає за заземлення.

2.4.3. Датчик контролю якості повітря CCS811 + HDC1080 (CO2+VOCs).

CCS811 - це низько-потужний цифрового газовий датчик який інтегрує датчик газу для оксиду металу (MOX) для виявлення широкого асортименту летких органічних сполук (ЛОС) для контролю якості повітря в приміщенні за допомогою мікроконтролерного блоку (MCU), який включає аналого-цифровий перетворювач (АЦП) та I²C інтерфейс. CCS811 заснований на унікальній технології мікроконфорки AMS, що дозволяє отримати дуже надійне рішення для газових датчиків, дуже швидкий час циклу та значне зниження середньої потужності споживання. Інтегрований

мікроконтроллер управляє режимами роботи датчика та вихідними даними датчиків. Цифровий I²C інтерфейс значно спрощує апаратне та програмне забезпечення. CCS811 підтримує інтелектуальні алгоритми для обробки необробленого датчика вимірювання для виведення значення TVOC або еквівалентного CO2 (eCO2). CCS811 підтримує кілька режимів вимірювання, які були оптимізовані для споживання низької потужності під час роботи датчика в режимі вимірювання та в режимі очікування, що подовжує термін служби акумулятора в портативних пристроях.

Переваги датчика CCS811+ HDC1080:

- керує режимами і вимірюваннями датчика при виявленні VOC;
- забезпечує рівень eCO2 індикації TVOC;
- простий в апаратній та програмній інтеграції;
- підходить для конструкцій малого форм-фактора;
- розроблений для великого обсягу прийому даних та надійності (термін служби більше 5 років).

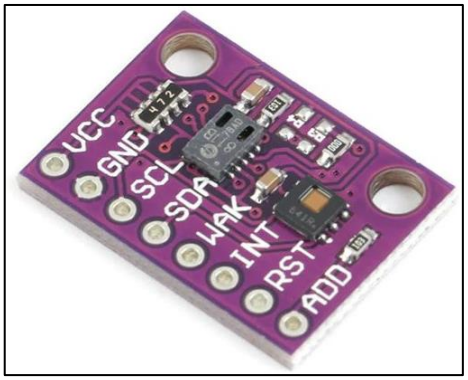

Рис. 2.8. Датчик CCS811 + HDC1080

2.5. Висновки до розділу.

Для розробки комп’ютерного пристрою контролю якості повітря було використано три різних датчика, які вимірюють різні параметри: температуру та вологість, частинки пилу, CO2 та TVOC.

Для розробки ескізів було використано мову програмування C++. Вона дозволяє розробнику вручну контролювати витрати пам’яті, є надшвидкою в порівнянні з іншими мовами програмування та є вдосконаленою версією мови програмування C.

Для розробки ескізів було обрано середовище розробки Arduino IDE. Воно є рідним для плат розробки Arduino, оптимізоване для роботи з ними. В самому середовищі розробки є приклади програм, для того, щоб ознайомитись з основними функції плат розробки.

РОЗДІЛ 3
РОЗРОБКА КОМП'ЮТЕРНОГО ПРИСТРОЮ КОНТРОЛЮ ЯКОСТІ ПОВІТРЯ.

3.1. Налаштування та підключення датчика температури та вологості DH11.

Перш за все, нам необхідно знати, які роз'єми ми будемо використовувати на нашій платі розробки Arduino UNO. Для цього існує офіційна документація, розміщена на сайті виробника. Arduino UNO має 7 роз'ємів, які відповідають за напругу, 6 аналогових роз'ємів та 19 цифрових роз'ємів.

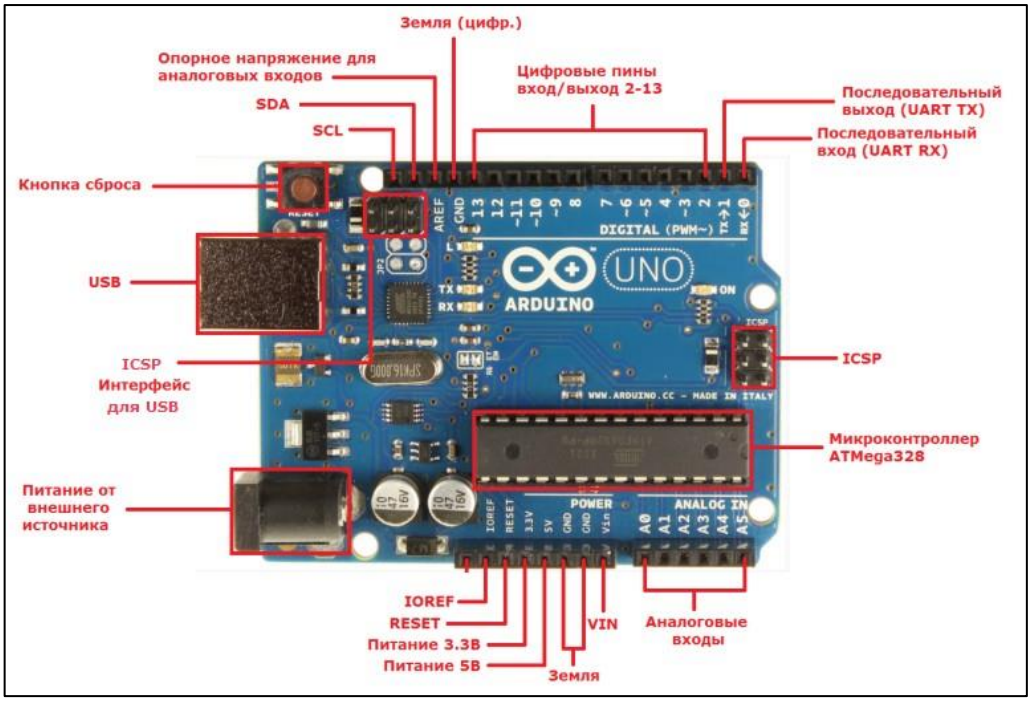

Рис. 3.1. Опис роз'ємів плати Arduino UNO

Кафедра КІТ (47)				НАУ 21 06 91 000 ПЗ			
Виконав	Герус Р.О.			ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОГО ПРИСТРОЮ КОНТРОЛЮ СТАНУ ЗАБРУДНЕННЯ ПОВІТРЯ	*Літера*	*Аркуш*	*Аркушів*
Керівник	Зіатдінов Ю.К.					39	11
Кон-					411 122		
Н-котрол.	Шевченко О.П.						
Зав. каф.	Савченко А.С.						

Для датчика DH11 ми будемо використовувати 3 роз’єма, з яких один буде поданий на живлення 5 В, один поданий на цифровий роз’єм D2, а останній на землю. Після підключення датчика, плата буде мати наступний вигляд:

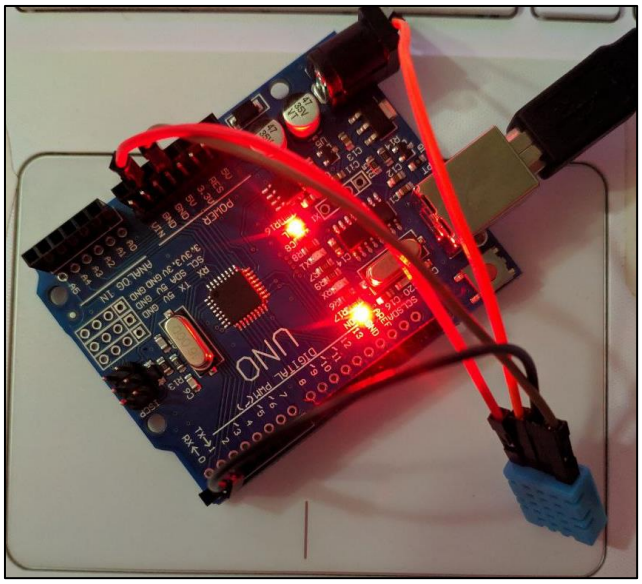

Рис 3.2. Вигляд плати розробки після підключення датчика DH11

3.1.1 Реалізація програми зчитування даних з датчика DH11.

Створимо новий ескіз. Для цього перейдемо у середовище розробки Arduino IDE та виберемо відповідне меню. Arduino IDE генерує функції setup() та loop() за замовчуванням, оскільки вони є основними в роботі з платою розробки.

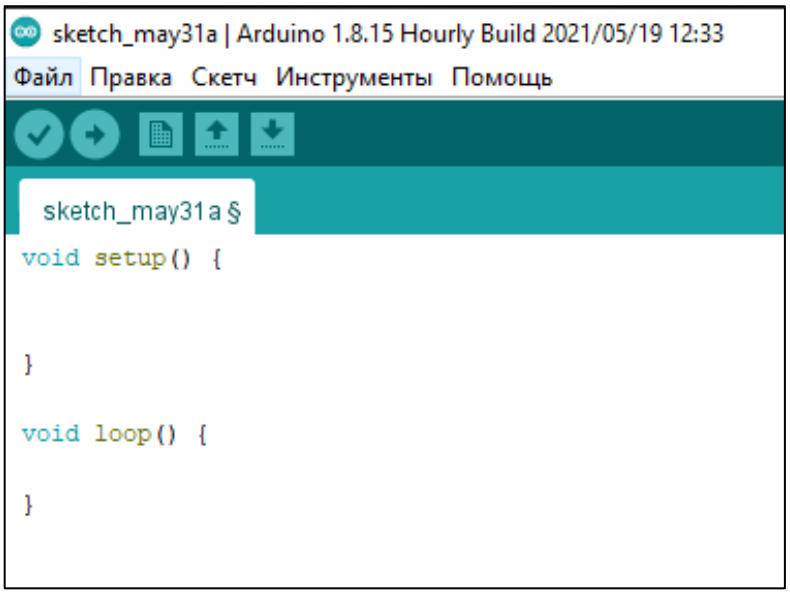

Рис 3.3. Створення нового ескізу

Далі, нам необхідно включити необхідну бібліотеку для роботи з датчиком DH11. Для цього в мові C++ використовується ключове слово #include. Так ми даємо препроцесору зрозуміти, що перед компіляцією програми, йому необхідно знайти файл заголовку бібліотеки та включити його в проект. Процес включення окремих файлів в проект аналогічний тому, якби у нашому основному файлі з ескізом був код, необхідний для коректної взаємодії датчика з платою. Наступним кроком стане оголошення макросу. В мові C++ це робиться за допомогою ключового слова #define. Директива препроцесора #define створює символічні константи. Символічна константа називається макросом. Коли цей рядок з'являється у файлі, усі подальші входження макросів у цьому файлі заміняться текстом заміни перед компіляцією програми.

Наступним важливим кроком стане реалізація функції setup(). Нам необхідно передати значення параметру швидкості зчитування даних. Це говорить Arduino готуватися до обміну повідомленнями з послідовним монітором зі швидкістю передачі даних 9600 біт в секунду. Це 9600 двійкових одиниць або нулів в секунду, і це зазвичай називають швидкістю передачі.

```
void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 test!"));

  dht.begin();
}
```

Рис. 3.4. Ініціалізація та реалізація функції setup()

Після того, як ми реалізували функцію setup(), переходимо до реалізації другої основної функції, loop(). Першим ділом, нам необхідно встановити затримку між збором замірів. Зчитування температури або вологості займає близько 250 мс. Після того, як затримка вказана, оголошуємо змінні для зберігання наших даних. Змінні обов'язково повинні бути типу float, тобто число з плаваючою комою. Це означає, що ми будемо отримувати точні дані про температуру та вологість. Обов'язковим фактором у написанні ескізу для роботи з датчиком є перевірка роботи на помилки. Тобто, якщо датчик не встиг зчитати дані, або дані не доступні, ми повинні це бачити на екрані виводу! Також, я вирішив додати обчислення індексу теплоти. Це також робиться за допомогою оголошення змінних та виклику методу, який це робить. Після того, як ми оголосили всі необхідні нам змінні та викликали відповідні методи, ми можемо налаштовувати вивід зібраних нами даних на екран виводу. Для цього існує стандартний метод, під назвою print(). Ми передаємо туди аргументом наші змінні, в яких зберігаються дані, а метод print() друкує нам їх на екран виводу.

```ino
#include "DHT.h"

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 увімкнено!"));

  dht.begin();
}

void loop() {
  // Затримка між збороом замірів.
  delay(2000);

  float h = dht.readHumidity(); // Зчитуємо дані про вологість
  float t = dht.readTemperature();// Зчитуємо дані про температуру
  float f = dht.readTemperature(true); // Задаємо логічний прапорець, щоб дані надходили у Фаренгейтах

  // Перевірка на момилки зчитування
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Помилка зчитування з датчика DH11!"));
    return;
  }

  // Обчислення індексу теплоти в Фаренгейтах
  float hif = dht.computeHeatIndex(f, h);
  // ООбчислення індексу теплоти в градусаз по Цельсію
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Вологість: "));
  Serial.print(h);
  Serial.print(F("%  Температура: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Індекс теплоти: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

Рис. 3.5. Остаточний вигляд ескізу для роботи з датчиком DH11

Тепер, завантажимо наш ескіз на плату Arduino UNO та протестуємо його в роботі. Для цього натиснемо відповідну кнопку з основних 6 кнопок для роботи з платою. Після успішної компіляції ескізу та завантаження, відкриємо екран виводу, та будемо спостерігати за даними, які приходять з датчика.

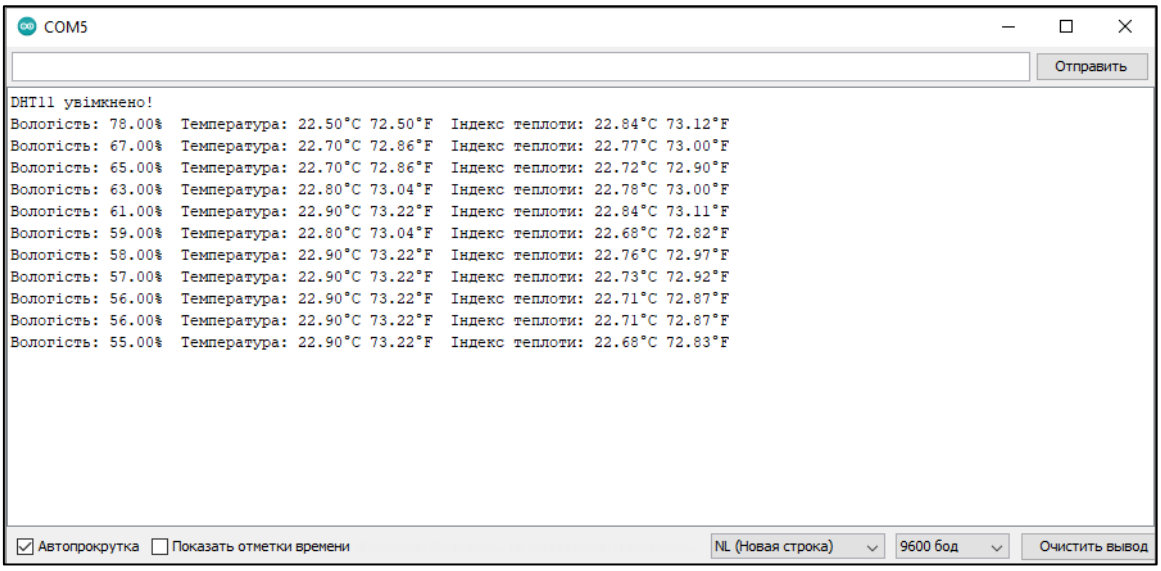

Рис. 3.6. Вивід результатів замірів з датчика DH11

Щоб пересвідчитись у тому, що дані приходять коректно, симулюємо зміну температури та вологості в кімнаті. Для цього можна використати будь який пристрій для нагрівання повітря, наприклад фен. На рис. 3.7 зображені результати роботи датчика до увімкнення фену та після. Як видно, датчик працює коректно.

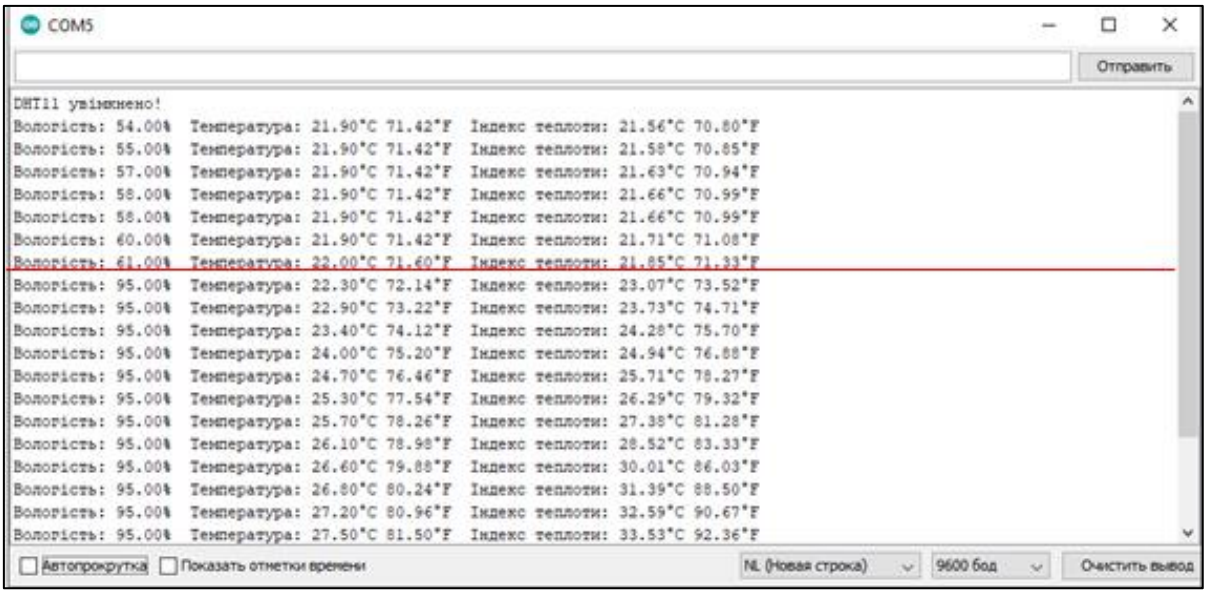

Рис. 3.7 Результат роботи при різкій зміні температури та вологості

3.2. Налаштування та підключення датчика вимірювання пилу у повітрі PMS1003.

Аналогічно з датчиком DH11, ми повинні знати необхідні нам для роботи роз’єми. Датчик PMS1003 використовує інший протокол передачі даних, а саме UART. Ключова відмінність він звичайного цифрового протоколу полягає в тому, що для отримання а передачі даних ми будемо використовувати відразу два роз’єми, на відміну від одного в цифровому протоколі. Також нам необхідно підключити його до землі та подати напругу на датчик, за допомогою роз’єму живлення 3.3 В. Саме на такій напрузі працює даний датчик. Після підключення датчика, плата буде мати такий вигляд:

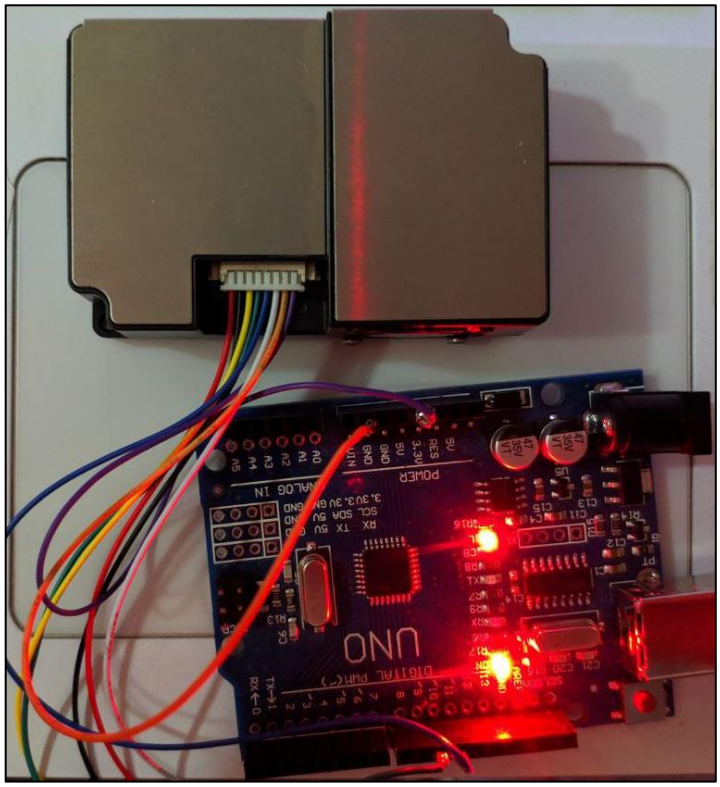

Рис. 3.8. Вигляд плати розробки після підключення датчика PMS1003

3.2.1. Реалізація програми зчитування даних з датчика PMS1003.

Аналогічно з попереднім підрозділом, створюємо новий ескіз. На відміну від попереднього ескізу, нам необхідно створити буфер, куди будуть записуватись отримані дані. Оскільки розмір даних, що нам приходять дорівнює 32 байти, відповідно буфер буде аналогічного розміру. Далі, нам необхідно зазначити, що контакти датчика, які відповідають за отримання та передачу даних, тобто RX та TX, підключені до 10 та 11 цифрового роз'єму відповідно. В функції setup() ініціалізуємо роботу датчика та встановлюємо затримку. В функції loop() ми, як вже було зазначено, обов'язково робимо перевірки, тому перевіримо чи співпадає розмір буферу з заданим нами значенням. На відміну від датчика DHT11, нам необхідно оголосити додаткові функції для перевірки отриманого нами значення та передачі даних на комп'ютер.

Особливими є функції передачі даних на комп'ютер. В них, для коректного відображення нам даних використовується оператор бітового зсуву <<. Це необхідно для того, щоб взяти дані саме з заміряними даними, оскільки датчик віддає дані про адресу, на якій він працює, свій статус і т.д.

```
//передаємо значення PM на ПК
int transmitPM01(unsigned char *thebuf)
{
  int PM01Val;
  PM01Val=((thebuf[3]<<8) + thebuf[4]); //підрахуємо значення PM1.0 модуля повітряного датчика
  return PM01Val;
}

//передаємо значення PM на ПК
int transmitPM2_5(unsigned char *thebuf)
{
  int PM2_5Val;
  PM2_5Val=((thebuf[5]<<8) + thebuf[6]);//підрахуємо значення PM2.5 модуля повітряного датчика
  }

//передаємо значення PM на ПК
int transmitPM10(unsigned char *thebuf)
{
  int PM10Val;
  PM10Val=((thebuf[7]<<8) + thebuf[8]); //підрахуємо значення PM10 модуля повітряного датчика
  return PM10Val;
}
```

Рис.3.9. Реалізація функцій передачі даних на комп'ютер

Завантажимо наш ескіз на плату розробки та протестуємо його в роботі. Як видно з рис. 3.10. Дані приходять коректно, показники вмісту пилу змінюються динамічно.

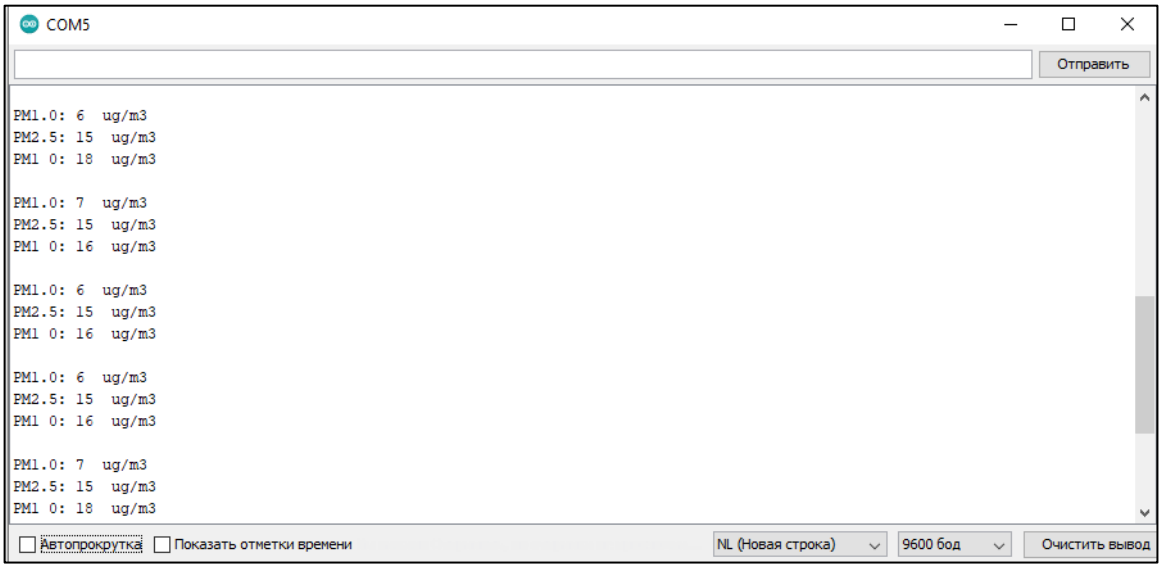

Рис. 3.10. Вивід результатів роботи датчика PMS1003

3.3. Налаштування та підключення датчика контролю якості повітря CCS811 + HDC1080.

По аналогії з попередніми датчиками, визначаємо необхідні нам роз'єми для підключення датчика CCS811 + HDC1080. Він працює за допомогою протоколу I2C, тому нам необхідно використати роз’єми SCL та SDA для отримання та передачі даних. Цей датчик працює від напруги 5 В, та аналогічно потребує заземлення. Після підключення датчика до плати, він буде мати наступний вигляд:

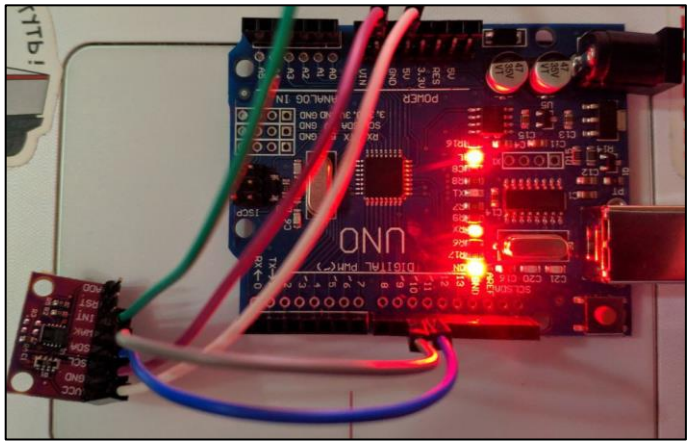

Рис. 3.11. Вигляд плати розробки після підключення датчика CCS811 + HDC1080

3.3.1. Реалізація програми зчитування даних з датчика CCS811 + HDC1080.

Для роботи з цим датчиком, нам необхідні дві бібліотеки. Це бібліотека, яка відповідає за реалізацію протоколу даних I2C та бібліотека самого датчика. Також, нам необхідні зазначити, на якій адресі працює цей датчик. За замовчуванням це 0x5A. Функція setup() буде відрізнятись лише тим, що ми ініціалізуємо обладнання для I2C. В функції loop() ми перевіряємо, чи готові дані для їх зчитування, та передаємо їх в метод, який відповідальний за обробку отриманих даних та їх передачу на комп'ютер.

```
#include <Wire.h>

#include "SparkFunCCS811.h"

#define CCS811_ADDR 0x5A //I2C адреса

CCS811 mySensor(CCS811_ADDR);

void setup()
{
  Serial.begin(115200);
  Serial.println("CCS811 Basic Example");

  Wire.begin(); //Ініціалізація обладнання I2C

  if (mySensor.begin() == false)
  {
    Serial.print("Помилка CS811. Будь ласка, перевірте коректність підключення. Перехід в режим очікування...");
    while (1)
      ;
  }
}

void loop()
{
  //Перевірка, чи готові дані за допомогою .dataAvailable ()
  if (mySensor.dataAvailable())
  {
    //Якщо так, просимо датчик прочитати та обчислити результати.
    //Отримуємо їх пізніше
    mySensor.readAlgorithmResults();

    Serial.print("CO2[");
    //Повертає обчислене значення CO2
    Serial.print(mySensor.getCO2());
    Serial.print("] tVOC[");
    //Повертає розраховане значення TVOC
    Serial.print(mySensor.getTVOC());
    Serial.print("] millis[");
    //Відобразити час з моменту запуску програми
    Serial.print(millis());
    Serial.print("]");
    Serial.println();
  }

  delay(10); //Не засмічуємо шину I2C
}
```

Рис. 3.12. Кінцевий вигляд ескізу для роботи з датчиком CCS811 + HDC1080

Завантажимо наш ескіз на плату розробки та протестуємо його в роботі. Виходячи з даних, які зображені на рис. 3.13. Дані приходять коректно, вони є динамічними, що свідчить про те, що датчик працює коректно.

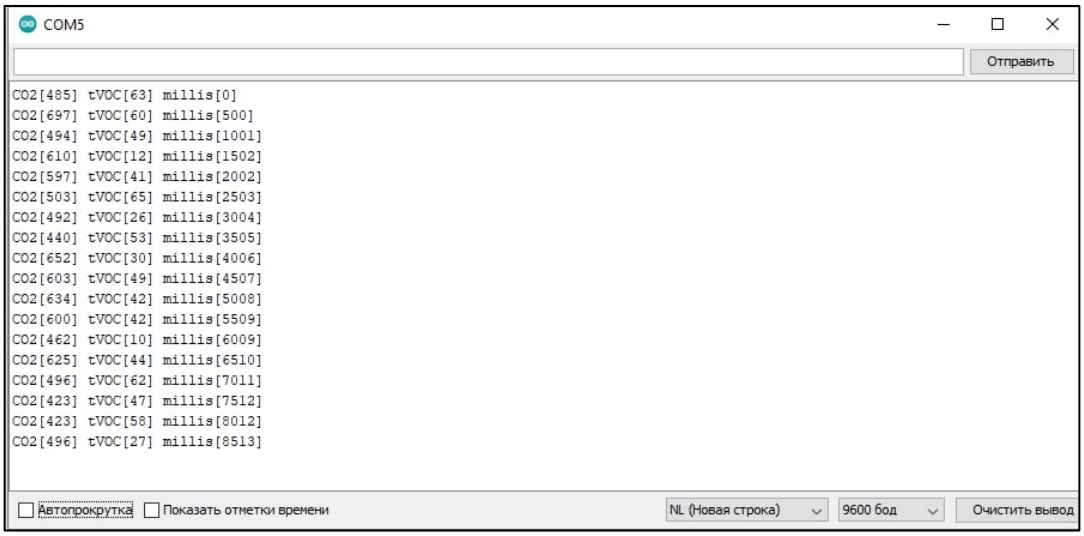

Рис. 3.13. Результати роботи датчика CCS811 + HDC1080

3.4. Висновки до розділу.

Основними етапами при розробці комп'ютерного пристрою контролю якості повітря є налаштування та підключення кожного датчика до плати, дослідження розміщення необхідних контактів та написання ескізів, для відображення замірів з кожного датчика. Для налаштування та підключення кожного датчика були створені та реалізовані наступні функції та макроси:

- стандартні функції setup() та loop();
- додаткові функції для датчика пилу PMS1003 transmitPMx();
- макроси з заданою адресою для датчика CCS811 + HDC1080 та довжиною буфера для PMS1003.

В результаті був створений комп'ютерний пристрій для замірювання різних покажчиків забруднення повітря, який може використовувати кожна людина в жилому приміщенні або офісі, в залежності від потреб.

ВИСНОВКИ

В процесі дослідження теми забруднення повітря в приміщеннях, стало зрозуміло, що це може бути великою проблемою для нашого здоров’я. Деякі люди ніколи й не знали про частинки пилу, які за розміром більш тонкі, ніж людська волосина, або про те, як відносна вологість може впливати на їх здоров’я. Не менш важливим фактором є вміст CO2 та TVOC, які формуються від продуктів життєдіяльності людини.

Постійний контроль за якістю повітря – це необхідність, яку повинне зрозуміти людство, для того щоб запобігти багатьом захворюванням, які можуть початись банально з того, чим ми дихаємо.

При розгляді сучасних систем контролю якості повітря було виявлено великий недолік, а саме те, що такі системи використовується лише в глобальних масштабах, і необхідні більш для загального контролю кількості викидів бруду в повітря.

В процесі розробки я використав наступні технології:

- мова програмування C++. Сучасна мова програмування яка довела свою швидкодію та універсальність;
- arduino UNO. Це плата розробки для початківців але з широкими можливостями зважаючи на те, що існує велика спільнота, яка може допомогти та поділитись своїми знаннями;
- arduino IDE. Середовище розробки, призначене спеціально для плат Arduino, оптимізоване для роботи та написання ескізів;

Основними можливостями розробленого комп’ютерного пристрою контролю якості повітря є:

- вимірювання температури, вологості та теплового індексу;
- вимірювання значень CO2 та TVOC;
- вимірювання значень забрудненості повітря пиловими частками PM1.0, PM2.5, PM10.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bjarne Stroustrup The C++ Programming Language (4th Edition): у 4 т. / Bjarne Stroustrup; Вид-во Addison-Wesley Professional, 2013. – 1368 с.
2. Programming Arduino: Getting Started With Sketches (second edition): у 2 т. / Simon Monk; Вид-во McGraw-Hill Education TAB, 2016. – 191 с.
3. Arduino Cookbook / Michael Margolis, Brian Jepson, Nicholas Robert Weldin; Вид-во O'Reilly and Associates, 2012. – 721 с.
4. Arduino: Advanced Methods and Strategies of Using Arduino / Ethan Thorpe; Вид-во Ethan Thorpe, 2020. – 160 с.
5. Arduino Programming: The Ultimate Guide For Making The Best Of Your Arduino Programming Projects / Damon Parker, Sean Antony; Вид-во New Begin Ltd, 2020. – 230 с.
6. Microcontroller and Smart Home Networks / Dawoud Shenouda Dawoud, Peter Dawoud; Вид-во River Publishers, 2020. – 350 с.
7. ARM-Based Microcontroller Multitasking Projects / Dogan Ibrahim; Вид-во Newnes, 2020. – 524 с.
8. Вікіпедія. Вільна енциклопедія. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Arduino (дата звернення 25.05.2021) - Назва з екрана.

ДОДАТКИ

Додаток А

Програмна реалізація файлу DHT11_sensor.ino

```
#include "DHT.h"

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 увімкнено!"));

  dht.begin();
}

void loop() {
  // Затримка між збороом замірів.
  delay(2000);

  float h = dht.readHumidity(); // Зчитуємо дані про вологість
  float t = dht.readTemperature();// Зчитуємо дані про температуру
  float f = dht.readTemperature(true); // Задаємо логічний прапорець, щоб дані надходили у Фаренгейтах

  // Перевірка на момилки зчитування
  if (isnan(h) || isnan(t) || isnan(f)) {
```

```
    Serial.println(F("Помилка зчитування з датчика DH11!"));
    return;
}

  // Обчислення індексу теплоти в Фаренгейтах
  float hif = dht.computeHeatIndex(f, h);
  // Обчислення індексу теплоти в градусаз по Цельсію
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print(F("Вологість: "));
  Serial.print(h);
  Serial.print(F("%  Температура: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Індекс теплоти: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

Додаток Б

Програмна реалізація файлу PM.ino

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#define LENG 31   //0x42 + 31 байт, що дорівнює 32 байтам
unsigned char buf[LENG];

int PM01Value=0;          //визначаємо значення PM1.0 модуля повітряного датчика
int PM2_5Value=0;         //визначаємо значення PM2.5 модуля повітряного датчика
int PM10Value=0;         //визначаємо значення PM10 модуля повітряного датчика

SoftwareSerial PMSerial(10, 11); // RX, TX

void setup()
{
  PMSerial.begin(9600);
  PMSerial.setTimeout(1500);
  Serial.begin(9600);
}

void loop()
{
  if(PMSerial.find(0x42)){
    PMSerial.readBytes(buf,LENG);

    if(buf[0] == 0x4d){
      if(checkValue(buf,LENG)){
        PM01Value=transmitPM01(buf); //підрахуємо значення PM1.0 модуля повітря-
ного датчика.
```

```
PM2_5Value=transmitPM2_5(buf);//підрахуємо значення PM2.5 модуля повітряного датчика
      PM10Value=transmitPM10(buf); //підрахуємо значення PM10 модуля повітряного датчика
     }
   }
  }

  static unsigned long OledTimer=millis();
   if (millis() - OledTimer >=1000)
   {
    OledTimer=millis();

    Serial.print("PM1.0: ");
    Serial.print(PM01Value);
    Serial.println("  ug/m3");

    Serial.print("PM2.5: ");
    Serial.print(PM2_5Value);
    Serial.println("  ug/m3");

    Serial.print("PM1 0: ");
    Serial.print(PM10Value);
    Serial.println("  ug/m3");
    Serial.println();
   }

}
char checkValue(unsigned char *thebuf, char leng)
```

```
{
  char receiveflag=0;
  int receiveSum=0;

  for(int i=0; i<(leng-2); i++){
  receiveSum=receiveSum+thebuf[i];
  }
  receiveSum=receiveSum + 0x42;

  if(receiveSum == ((thebuf[leng-2]<<8)+thebuf[leng-1]))  //перевіряємо серійні дані
  {
    receiveSum = 0;
    receiveflag = 1;
  }
  return receiveflag;
}

//передаємо значення PM на ПК
int transmitPM01(unsigned char *thebuf)
{
  int PM01Val;
  PM01Val=((thebuf[3]<<8) + thebuf[4]); //підрахуємо значення PM1.0 модуля повіт-
ряного датчика
  return PM01Val;
}

//передаємо значення PM на ПК
int transmitPM2_5(unsigned char *thebuf)
{
```

```
  int PM2_5Val;
  PM2_5Val=((thebuf[5]<<8) + thebuf[6]);//підрахуємо значення PM2.5 модуля повітряного датчика
  }

//передаємо значення PM на ПК
int transmitPM10(unsigned char *thebuf)
{
  int PM10Val;
  PM10Val=((thebuf[7]<<8) + thebuf[8]); //підрахуємо значення PM10 модуля повітряного датчика
  return PM10Val;
}
```

Додаток В

Програмна реалізація файлу CCS811.ino

```
#include <Wire.h>

#include "SparkFunCCS811.h"

#define CCS811_ADDR 0x5A //I2C адреса

CCS811 mySensor(CCS811_ADDR);

void setup()
{
  Serial.begin(115200);
  Serial.println("CCS811 Basic Example");

  Wire.begin(); //Ініціалізація обладнання I2C

  if (mySensor.begin() == false)
  {
    Serial.print("Помилка CS811. Будь ласка, перевірте коректність підключення. Перехід в режим очікування...");
    while (1)
      ;
  }
}

void loop()
{
  //Перевірка, чи готові дані за допомогою .dataAvailable ()
```

```
  if (mySensor.dataAvailable())
{
    //Якщо так, просимо датчик прочитати та обчислити результати.
    //Отримуємо їх пізніше
    mySensor.readAlgorithmResults();

    Serial.print("CO2[");
    //Повертає обчислене значення CO2
    Serial.print(mySensor.getCO2());
    Serial.print("] tVOC[");
    //Повертає розраховане значення TVOC
    Serial.print(mySensor.getTVOC());
    Serial.print("] millis[");
    //Відобразити час з моменту запуску програми
    Serial.print(millis());
    Serial.print("]");
    Serial.println();
  }

  delay(10); //Не засмічуємо шину I2C
}
```