Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра комп'ютерної інженерії та електроніки

Дзінько Владислав Миколайович
Dzinko Vladyslav

УДК 004:621.3

Спеціальність 123 «Комп'ютерна інженерія»

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавра

Розробка web-застосунку взаємодії мікроконтролера і клієнт-серверного комплексу

Development of web application of interaction of microcontroller with client-server complex

Науковий керівник:
Професор Когут І. Т.
Рецензент:
Професор Прокопів В.В.

Івано-Франківськ
2020

Формат	Поз.	Позначення	Найменування	К-ть	Прим.
А4			Схема підключення для прошивання	1	
А4			Схема підключення для взаємодії	1	
А4			Пояснювальна записка	75	

Змн.	Арк.	№ докум.	Підпис	Дата	123.КІ-41.3		
Розробив		Дзінько В.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Когут І.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

Тенденція до постійного використання людьми пристроїв з неперервним доступом до мережі Інтернет вимагає від розробників застосування до своїх виробів функціоналу безпровідного керування по Bluetooth або Wi-Fi, оскільки це значно полегшує доступ до інтерфейсу керування користувачем, та забирає необхідність у розробці окремого програмного забезпечення для цього, що дозволяє розробникам зекономити час та кошти при розробці. Веб-застосунок віддаленого керування значно знижує поріг входження в користування ним, оскільки прирівнюється, наприклад, до користування сайту з прогнозом погоди.

Основними завданнями при розробці цього програмно-апаратного комплексу були напрямлені на побудову цілісної архітектури застосунку та простого користувацького інтерфейсу. Це надало можливість показати такі його можливості, як зрозумілість людині(пересічному користувачу) інтерфейс та високу швидкодію. За допомогою веб-інтерфейсу, через браузер, використовуючи методи взаємодії з ним, здійснюється можливість віддаленого керування проектом на основі мікроконтролера ATMega.

Робота міститиме опис апаратної частини зі схемою та код програмного рішення розробленого за допомогою середовища програмування Ардуіно IDE. Зі сторони клієнта використовуватиметься HTML5, CSS3 та JavaScript.

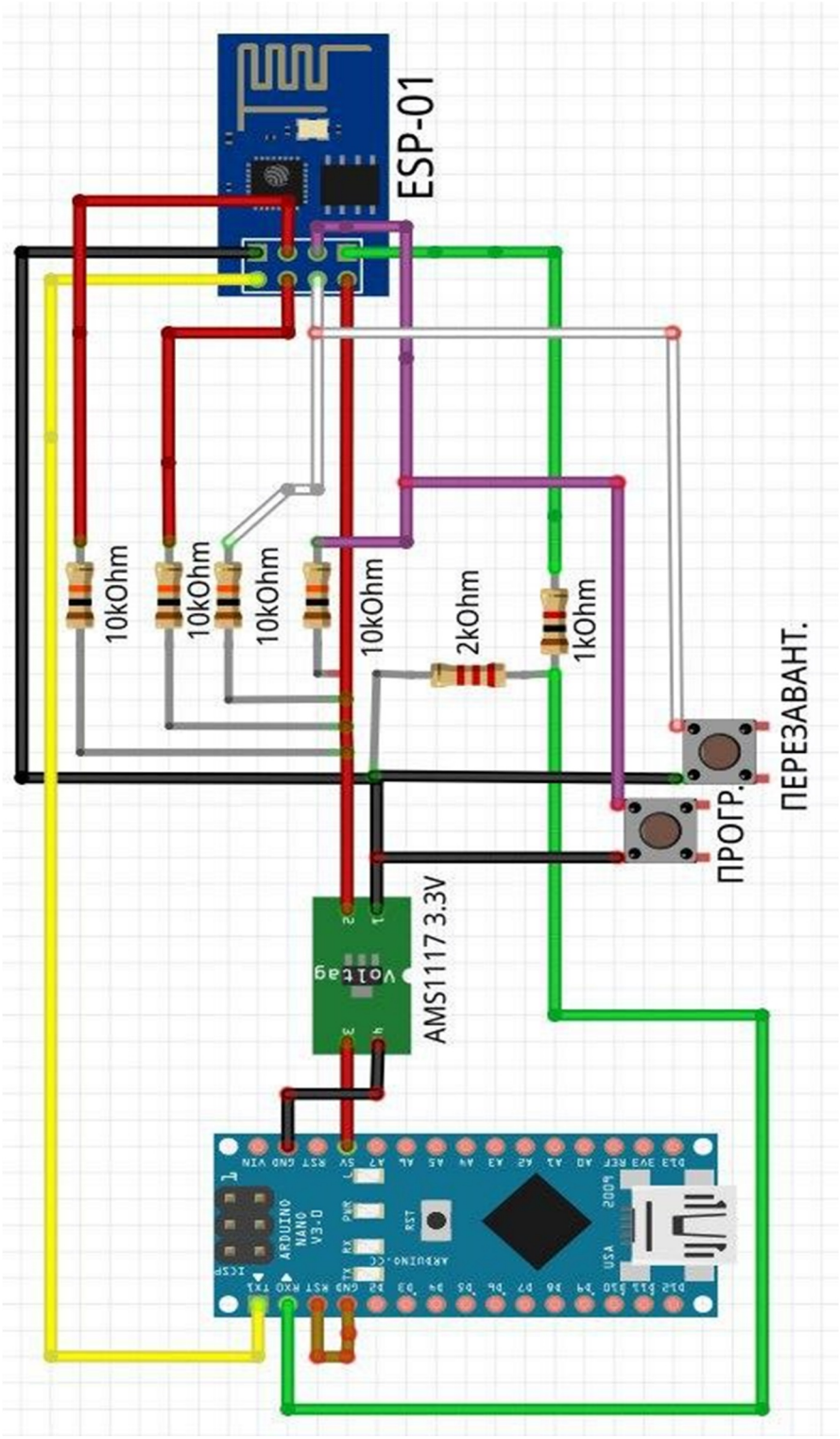
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Дзінько В.			Анотація	*Літ.*	*Арк.* 3 / *Аркушів* 1
Перевірив		Когут І.					
Н. Контр.							
Затвердив							

ABSTRACT

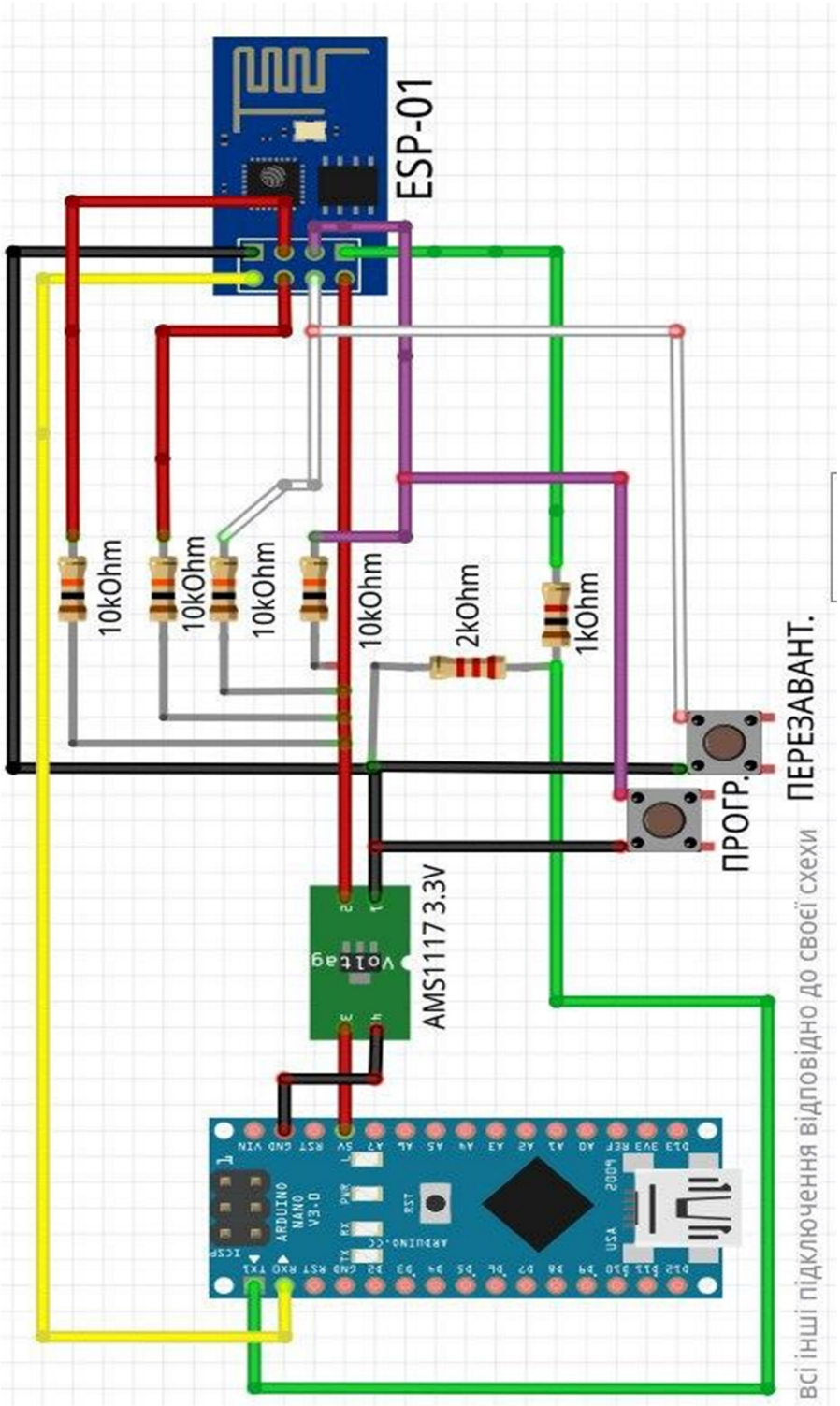
The tendency for people to constantly use devices with continuous Internet access requires developers to apply Bluetooth or Wi-Fi management features to their products, since it greatly simplifies access to the user management interface and eliminates the need to develop separate software for allows developers to save time and money on development. A web-based remote-control application significantly lowers the entry threshold for using it, because it equates, for example, to using a weather forecast site.

The main tasks in the development of this software and hardware complex were aimed at building a coherent application architecture and a simple user interface. This made it possible to show its capabilities such as human-readability (average user) interface and high speed. Using the web interface, through the browser, using methods of interaction with it, the possibility of remote project management based on the ATMega microcontroller.

The work will contain a description of the hardware with the scheme and the code of the software solution developed using the Arduino IDE. Client-side will use HTML5, CSS3 and JavaScript.



Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Дзінько В.			Схема підключення для прошивання	*Літ.*	*Арк.* 5 / *Аркушів* 1
Перевірив		Когут І.					
Н. Контр.							
Затвердив							



Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Дзінько В.			Схема підключення для взаємодії	*Літ.*	*Арк.* 6 / *Аркушів* 1
Перевірив		Когут І.					
Н. Контр.							
Затвердив							

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«Розробка web-застосунку взаємодії мікроконтролера і клієнт-серверного комплексу»

					123.КІ-41.3		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Дзінько В.			Пояснювальна записка	*Літ.*	*Арк.* 7 / *Аркушів* 75
Перевірив		Когут І.					
Н. Контр.							
Затвердив							

ЗМІСТ

ВСТУП

Інформація відграє значну роль в житті будь-якої людини. З початку часів існування людства ми отримували інформацію один від одного в різноманітних формах (у вигляді жестів, вигуків, слів). В подальшому з модернізацією суспільства невичерпним джерелом інформації став Інтернет.

Оскільки інтернет включає в себе десятки тисяч наукових, корпоративних, урядових та домашніх мереж, на сьогоднішній день, він є невід'ємною частиною систематизованої роботи, як багатьох організацій так і домашніх користувачів. Це призвело до потреби полегшення доступу користувачів до мережі Інтернет. Вирішенням цієї проблеми стала розробка технології Wi-Fi, яка надає можливість доступу до Інтернет мережі, що виключає фізичне підключення за допомогою кабелів.

Розумним пристроям в одній системі необхідне підключення до мережі, щоб надати змогу пристроям «спілкуватись» між собою та надати можливість користувачеві впливати на роботу цих пристроїв не за допомогою закидання нового програмного забезпечення в їх флеш-пам'ять та слідкувати за станом системи в цілому не на місці, а віддалено, через мережу Інтернет. В свою чергу для цього треба підключати відповідні мережеві модулі та тягнути кабелі до найближчого постачальника Інтернету. Wi-Fi значно спростить цю ситуацію, підключивши до розумних пристроїв потрібний модуль буде доступ до Інтернету, налаштувавши його, тоді буде можливість за допомогою улюбленого веб-браузера отримати доступ до всієї системи.

Підсистема бізнес-логіки містить реалізацію відповідної частини програми, яка не повинна цікавити користувача, але яка потрібна для роботи програми, а також механізм взаємодії з клієнтською частиною на основі HTTP-запитів. Інтерфейс користувача міститься у клієнтській частині, яка звертається до серверної частини через запити та отримує відповідь відповідного до свого запиту.

Розроблена серверна частина системи разом з клієнтською частиною складає єдину неподільну систему, основною метою якою є полегшення та прискорення віддаленого керування за допомогою стандарту IEEE 802.11(Wi-Fi) передачі цифрових потоків інформації по радіоканалах.

Розробка веб-застосунку для віддаленого безпровідного керування проектом під керуванням платою Ардуіно є потребою для спрощення доступу користувачем до інформації, якою володіє та керує плата Ардуіно. З економічної точки зору це зекономить кошти розробникам за рахунок відсутніх додаткових кабелів підключення плати керування до мережі Інтернет.

Безпровідний контроль за роботою мікроконтролера це корисна прикладна та складна задача, яка полегшить застосування систем на базі мікроконтролерів.

Метою роботи є створення web-застосунку взаємодії мікроконтролера і клієнт-серверного комплексу, для віддаленого керування.

Об'єкт дослідження – web-розробка із взаємодією мікроконтролера на платі Ардуіно Нано.

Предмет дослідження – створення web-застосунку взаємодії мікроконтролера і клієнт-серверного комплексу.

Для досягнення поставленої мети в роботі визначені і розв’язані такі **завдання**:

1. Розробка клієнт-серверного комплексу виконаного на мікроконтролері та його взаємодія з web-застосунком.

2. Реалізація даного проекту для віддаленого керування клієнт-серверним комплексом.

3. Налаштовування протоколу зв'язку між сервером та клієнтом.

4. Розробка інтерфейсу користувача для веб-сторінки.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ АРХІТЕКТУР

1.1. Визначення

Правильно розроблена інформаційна система опирається на цілісну основу, яка підтримує можливість зміни і, таким чином, вищу швидкість організації по мірі виникнення нових бізнес чи адміністративних ініціатив. Фонд, відомий як архітектура технологічної системи, опирається на основні телекомунікаційні мережі, бази даних та сховища даних, програмне забезпечення, устаткування та процедури, якими керують різні технічні спеціалісти. При масштабуванні справ архітектура організації досить часто пересікає національні кордони. Продумування і підтримка такої складної архітектури потребує обширного попереднього планування та послідовного ретельного впровадження для вирішення стратегічних корпоративних ініціатив, перетворень, різних злиттів та поглинань. Інфраструктура інформаційної системи повинна бути створена з урахуванням різноманітних варіацій майбутнього корпоративного розвитку[6].

Коли вони організовані в певні, специфічні інформаційні системи, що підтримують операції управління та систематизацію знань, тоді вони становлять архітектуру системи організації. Очевидно, що при розробці інфраструктури та архітектури інформаційної системи необхідно враховувати довгострокові загальні стратегічні плани організації.

Архітектура клієнт-сервер - це поділена будова застосунку, яка розподіляє завдання або робочі навантаження між надавачами ресурсів або послуг, що іменуються серверами, і отримувачами послуг, що іменуються клієнтами. Майже завжди клієнти з серверами обмінюються повідомленнями через комп'ютерну мережу на різному устаткуванні, передбачений і такий варіант, що клієнт з сервером можуть існувати в одній системі. Адміністратор серверної машини запускає серверний додаток, який ділиться своїми ресурсами з клієнтами, за допомогою певних мережних протоколів. Клієнт не ділиться жодним із своїх ресурсів з сервером, але він вимагає вміст чи послуги від сервера. Клієнт «відкриває» сеанс зв'язку із сервером, який в свою чергу очікує на вхідний запит, який в свою чергу має обробити. Прикладами застосунків, які застосували клієнт-

серверну модель, є FTP-клієнти(FileTransferProtocol - Протокол передачі файлів), глобальна мережа Інтернет, будь-який веб-сайт, друк по мережі та інші.

Під час обчислень сервер - це застосунок або набір устаткування, що надає набір функціоналу для застосунків, яким він потрібен, чи машин, які іменуються "клієнтами". Сервер часто може надавати різний функціонал, який інколи називається "послугами", наприклад, обмін інформацією або файлами між парою клієнтів чи виконати обчислення для клієнта. Єдиний серверна машина чи застосунок може надавати так звані «послуги» декільком клієнтам, в свою чергу одна клієнтська машина чи застосунок може використовувати набір серверів. Клієнтська програма може працювати на тій самій машині, що і сервер, також має змогу під’єднуватись до серверної програми через певну(глобальну, локальну) мережу на іншій машині. Звичайними серверними програмами є сервери роботи з базами даних, сервери файлів, сервери опрацювання інформації(яку машині клієнта довготривало опрацьовувати), сервери поштових служб, сервери мережевого друку, сервери веб-сайтів, сервери ігор і сервери застосунків.

Архітектура клієнт-сервер досить часто імплементується (майже завжди заміняється) як модель запиту-відповіді: машина клієнта надсилає свій запит у серверну програму, яка опрацьовує інформацію, і згідно ній відправляє відповідь клієнтській машині, зазвичай, у вигляді результату або підтвердженням про виконання. Маркування пристрою як устаткування серверного класу значить, що він спеціалізований для роботи як сервер. Це часто означає, що він є більш потужний і надійний, ніж прості комп'ютери користувачів, але також буває що, великі обчислювальні кластери(набір серверних машин) можуть складатися з багатьох, простих машин, змінних частин кластера[2].

Клієнт - це комплекс апаратного забезпечення з програмним забезпеченням для певного пристрою, який отримує доступ до інформації, наданої сервером. Сервер часто (але не завжди) знаходиться в іншій комп'ютерній мережі, і тоді клієнт отримує доступ до послуги через глобальну мережу.

Також можна сказати що, клієнт - це пристрій або програма, яка в рамках своєї роботи покладається на надсилання запиту в іншу програму або на комп'ютерне устаткування, яке отримує доступ до інформації, яка доступна серверу (який може бути, а може і не перебувати на іншій машині). Хороший приклад, веб-браузер - це клієнт, який підключається до веб-серверів і отримує веб-сторінки для відображення. Клієнти електронної пошти отримують електронну пошту з поштових серверів. Інтернет-чат використовує різноманітних клієнтів, які залежать від протоколу чату, який використовується. Багатокористувацькі відеоігри або онлайн-відеоігри можуть працювати як клієнти на кожному комп’ютері. Термін “клієнт” може також застосовуватися до комп'ютерів або пристроїв, на яких працює клієнтський застосунок, який відрізняється від серверного, або користувачів, які використовують програмне забезпечення клієнта(Рисунок 1.1.) [3].

Клієнт є частиною архітектури клієнт-сервер, яка широко використовується і сьогодні. Клієнти та сервери можуть бути комп'ютерними програмами, що працюють на одній машині та підключаються за допомогою між процесорних методів зв'язку. У поєднанні з Інтернет-сокетами програми можуть підключатися до процесу, що працює, можливо, в віддаленій мережі через набір протоколів Інтернет. Сервери чекають, коли потенційні клієнти зроблять запит, на який вони можуть відповісти[13].

Термін вперше застосували до пристроїв, які не здатні запускати власні автономні програми, але могли взаємодіяти з віддаленими комп'ютерами через мережу. Ці пристрої були клієнтами основного комп'ютера для обміну часу.

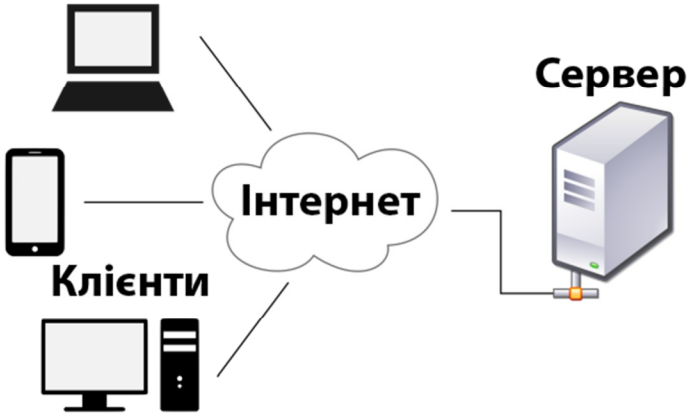


Рисунок 1.1. Мережа клієнтських комп'ютерів, що спілкуються з серверною машиною через мережу Інтернет.

1.2. Опис клієнт-серверної моделі

Модель «Клієнт-сервер» - це розшарована архітектура застосунку, яка розмежовує задачі та робоче навантаження між надавачами ресурсів або послуг, які іменуються серверами, і отримувачами послуг чи ресурсів, які іменуються клієнтами. Архітектура клієнт-серверного застосунку полягає у тому, що машина клієнта надсилає запит на серверні дані через певну мережу, сервер завжди очікує на запити, вміє їх обробляти і доставляє запитувану інформацію у вигляді пакетів даних назад клієнту. Клієнти не діляться жодними своїми ресурсами з серверами[4].

Як працює клієнт-серверна модель застосунку?

Клієнт: коли вимовляється слово Клієнт, це означає розмовляти з людиною або організацією, яка використовує певну послугу. Аналогічно в цифровому світі Клієнт - це пристрій, тобто здатний отримувати інформацію або використовувати певну послугу від надавачів(серверів) комп’ютер.

Сервери: Аналогічно, коли вимовляється слово "Сервери", це означає людину або середовище, які щось роблять. Аналогічно цьому в цифровому світі Сервер - це віддалений комп'ютер, який надає інформацію (дані) або доступ до певних послуг[15].

Отже, Клієнт в основному запитує щось, а Сервер обслуговує його потреби(запити), доки відповіді присутні у базі даних(Рисунок 1.2.1.).

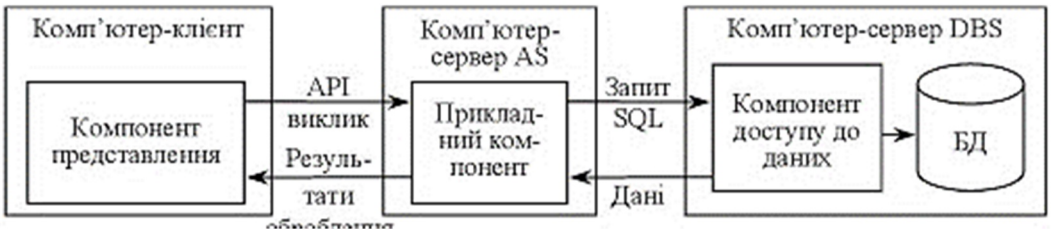


Рисунок 1.2.1. Клієнт-серверна модель.

Клієнти часто знаходяться на потужних комп'ютерах або на простих користувацьких пристроях, але може перебувати і на будь-яких пристроях, який має доступ до сервера, а сервери розташовані в інших локаціях певної мережі, зазвичай, на продуктивніших машинах, порівняно з клієнтськими, для швидкого отримання жаданої інформації клієнтом. Така модель опрацьовування інформації дуже ефективна, якщо клієнти та сервери мають свої завдання, які вони регулярно виконують. Сервер спеціально підготовлений для виконання потрібної операції. Наприклад, обробка даних в лікарні, клієнтський пристрій може запускати прикладну програму для введення інформації про пацієнтів, поки серверний комп'ютер працює в іншій програмі, яка керує базою даних, в якій інформація постійно оновлюється та зберігається. Багато клієнтів можуть получити відповідь від сервера практично одночасно, сервер розрахований на певне навантаження, яке він здатен обробити. Оскільки тепер і клієнтські, і серверні машини вважаються досить продуктивними пристроями, модель клієнт-сервер сильно відрізняється від старішої популярної моделі "мейнфрейм", в якій присутній основний комп'ютер з центральним ядром, яке виконувало усі задачі від «прив'язаних» до нього примітивних терміналів.

1.3. Класи клієнт–серверних додатків

Існує спектр різних реалізації, які по-різному ділять роботу між клієнтом і сервером. Обробку можна виділити кількома способами.

В рамках основної структури застосунків з архітектурою типу клієнт-сервер, опрацювання інформації може розподілятись між клієнтом і сервером по-різному.

Існує кілька основних видів обробки цікавої для користувача інформації. Це хост-обробка, серверна обробка, клієнтська обробка та спільна(кооперативна) обробка[18].

1.3.1. Обробка даних на базі хоста

Обробка запиту на стороні хоста - це надзвичайно велика кількість обчислень для серверу та відсутня обробка на стороні клієнта. Скоріше, обробка на основі хоста відноситься до традиційного основного середовища, в якому вся або практично вся обробка здійснюється на потужностях хоста. Часто користувальницький інтерфейс здійснюється через термінал. Якщо ж користувач використовує міні-комп'ютер або станцію доступу, як правило, обмежується роллю емулятора терміналу(Рисунок 1.3.1.).

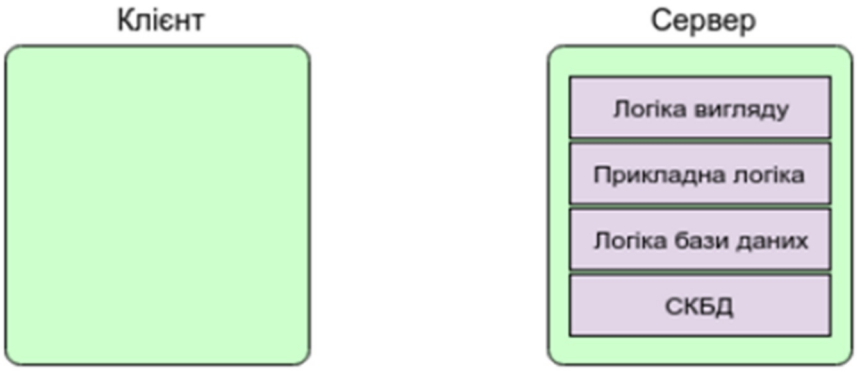


Рисунок 1.3.1. Обробка даних на базі хоста.

1.3.2. Обробка даних на базі сервера

Найбільш звичайним класом конфігурації сервера - це той, в якому клієнтська машина головним чином відповідає за надання графічного інтерфейсу користувача, тоді як практично вся обробка проводиться на сервері.

Обґрунтування такої конфігурації полягає в тому, що робоча станція користувача найкраще підходить для забезпечення зручного для користувача інтерфейсу, а бази даних та програми можуть легко підтримуватися в центральних системах на сервері. Незважаючи на те, що користувач отримує перевагу від кращого інтерфейсу, цей тип конфігурації, як правило, не піддається будь-яким значним приростам продуктивності, бо вся обробка лежить на сервері, або будь-яким фундаментальним змінам у фактичних бізнес-функціях, які підтримує система(Рисунок 1.3.2.).

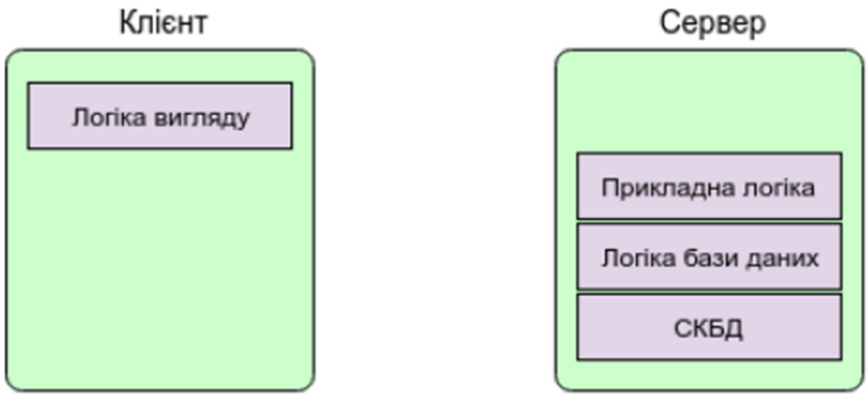


Рисунок 1.3.2. Обробка даних на базі сервера.

1.3.3. Обробка даних на базі клієнта

Цей вид архітектури клієнт-серверного застосунку показує протилежний підхід до опрацювання інформації: майже все опрацювання виконується на клієнтській машині, крім функціоналу по перевірці правильності інформації і іншого функціоналу, що пов’язаний з керуванням базами даних, який швидше та безпечніше виконувати серверною машиною. Ця схема виходить дуже складною для клієнтської машини, за рахунок більшого навантаження порівняно із іншими конфігураціями. Складні продуктивно-затратні логічні функції бази даних розміщуються на стороні клієнта. Перевірка інформації перед занесення у базу даних знаходяться у сервера, для безпеки та більшої швидкодії. По суті в цій

архітектурі сервер відповідає лише за доступ до бази даних[12]. Ця архітектура є чи не найпоширенішим підходом для клієнт-серверного комплексу в поточному використанні. Це дозволяє користувачеві використовувати застосунки, пристосовані до місцевих потреб(Рисунок 1.3.3.).

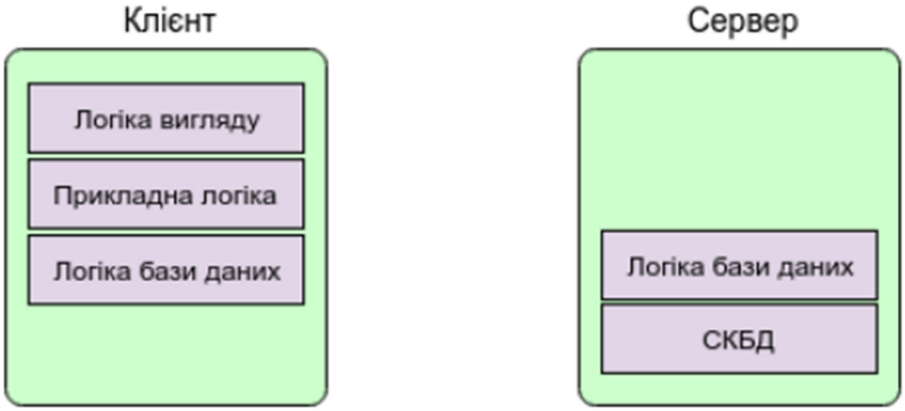


Рисунок 1.3.3. Обробка даних на базі клієнта.

1.3.4. Спільна обробка даних

У конфігурації спільної(кооперативної) обробки даних робота програми виконується оптимізовано, використовуючи переваги як клієнтських, так і серверних машин, а також використовується розподілення даних. Така конфігурація є більш складною для налаштування та обслуговування. Цей тип конфігурації може запропонувати більший приріст продуктивності клієнтської машини та більшу оптимізацію використання мережі, за допомогою якої під'єднується інший клієнт чи сервер.

Точне розділення даних та обробки застосунків буде залежати від характеру інформації в базі даних, типів підтримуваних застосунків, наявності сумісного устаткування надавача та моделей використання. Дана архітектура розрахована тільки на те що клієнт буде займатись лише інтерфейсом користувача та простою обробкою даних, щоб потужніша машина, сервер, обробила складніші дані швидше та повернула назад.

Серверні програми корисні, якщо ви хочете завантажувати з сервера або завантажувати файли з клієнтської машини на сервер(Рисунок 1.3.4.).

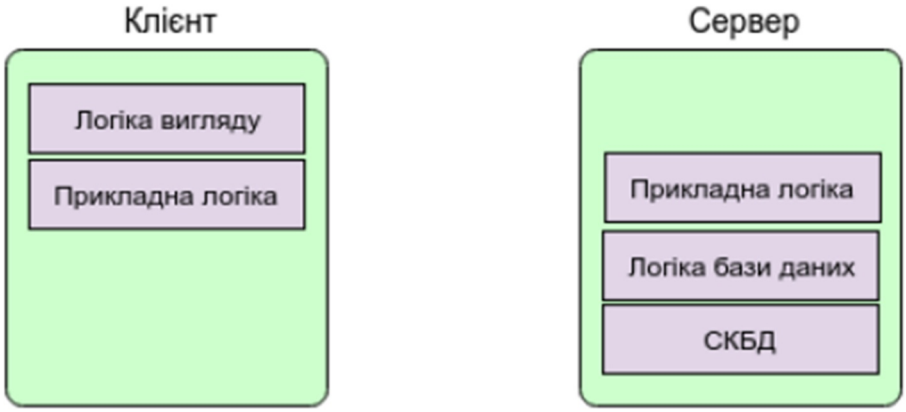


Рисунок 1.3.4. Спільна обробка даних.

РОЗДІЛ 2. МОДЕЛІ ВЗАЄМОДІЇ КЛІЄНТА І СЕРВЕРА

Два віддалені процеси додатку мають змогу спілкуватися зазвичай двома способами:

- Одноранговий: обидва віддалені процеси виконуються на одному рівні і вони обмінюються інформацією між собою, використовуючи певний спільний ресурс, як джерело даних.
- Клієнт-сервер: одна з двох програм працює як Клієнт і відправляє запити на доступ до певного ресурсу, а інша програма, що обробляє запити Клієнтів працює як Сервер.

У моделі розподіленої архітектури клієнт-серверного застосунку будь-яка програма може виступати як Сервер чи Клієнт. Серверна машина не є спеціальним видом машини, фізичним розміром машини чи обчислювальною потужністю, хоча зазвичай сервери є потужнішими, саме здатність обслуговувати клієнтські запити робить машину Сервером.

Система може виступати одночасно і як Сервер, і як Клієнт. Тобто, один процес на машині виконує роль Сервера, а інший процес виконує роль Клієнта та робить запити[11]. Також може бути такий варіант, що і клієнтські, і серверні процеси існують на тій самій машині.

Два процеси в моделі клієнт-сервер можуть взаємодіяти різними способами:

- Сокетами
- Віддаленими викликами процедур (RPC)

У цьому напрямі процес, що запущений як Сервер, відкриває запит, використовуючи спільний раніше обговорений порт і чекає, поки надійде якийсь запит від клієнта, який він може обробити, та повернути результат. Інший процес, що виконує роль Клієнта, відкриває запит, але замість того, щоб чекати на вхідний запит, клієнт створює ці запити.

Коли запит доходить від клієнта до сервера, цей запит обробляється. Це може бути і обміном інформацією, і запитом на ресурси, до яких має доступ сервер.

Інший спосіб використовує механізм коли один процес взаємодіє з іншим за допомогою віддаленого виклику процедур. Один процес, клієнтській машині викликає процедуру, що знаходиться на віддаленому хості. Процес віддаленого хоста, можна називати Сервером. Обом процесам виділяються потрібні ресурси. Це спілкування відбувається приблизно за таким алгоритмом:

- Клієнтська програма створює клієнтські запити. Вона передає далі всі потрібні параметри, що відносяться до локальної програми.
- Потім всі параметри упаковуються (маршальовано) і робиться системний виклик, щоб відправити їх на іншу сторону певної мережі.
- Ядро надсилає інформацію по мережі, а інший кінець, що її очікує, отримує цю інформацію.
- Віддалений хост передає ці дані в програмний елемент сервера, де їх потрібно використовувати.
- Параметри передаються до певної підпрограми, після чого виконується сама процедура опрацювання інформації.
- Результати за таким самим алгоритмом надсилаються клієнту у відповідь.

2.1. Веб-сервери

Веб-сервер - це серверний застосунок або певне устаткування, яке призначене для запуску цього застосунку, яке виконує обробку запитів клієнтів у глобальній мережі. В основному веб-сервер може містити один складний або багато нескладних веб-сайтів. Веб-сервер опрацьовує вхідні мережеві запити через HTTP та набір інших пов'язаних з HTTP протоколів.

У березні 1989 року пан Тім Бернерс-Лі надав можливість своєму роботодавцю ЦЕРН ознайомитись з новим, до того невідомим, проектом, метою якого було полегшити обмін інформацією між вченими по всьому світу за допомогою гіпертекстової системи.

В результаті роботи над проектом пан Бернерс-Лі випустив два додатки в 1990 році:

1. Веб-браузер під назвою WWW(WorldWideWeb - Всесвітня мережа);
2. Перший у світі веб-сервер, пізніше відомий як ЦЕРН httpd, який працював на комп'ютері NeXT з Операційною системою NeXTSTEP, потім цей веб-сервер було переписано і під інші ОС.

Протягом 1991 по 1994 роки відносна простота та ефективність початкових технологій, що використовуються для пошуку та обміну інформацією по всесвітній мережі, допомогли переписати їх для різних операційних систем та поширити їх використання серед наукових організацій та університетів, а з часом і по всій навчальній галузі.

У 1994 році пан Бернерс-Лі створив Всесвітній консорціум із питань глобальної Інтернет-мережі (W3C) для контролю подальшого розвитку багатьох використаних технологій (HTTP протокол, мова розмітки HTML та інших) методами опису та стандартизації.

Головними функціями веб-сервера повинні бути - збереження, опрацювання запитів та відправка веб-сторінок машинам клієнтів.

«Спілкування» клієнтської машини з серверною здійснюєтсья за допомогою протоколу передачі гіпертексту (HTTP – HyperTextTransferProtocol - Протокол передачі гіпертексту, пізніше HTTPS - HypertextTransferProtocolSecure - Захищений протокол передачі гіпертексту). Отримані сторінки - це зазвичай HTML-документи, до яких крім текстового вмісту можуть бути додані зображення та таблиці стилів(CSS-код - CascadingStyleSheets - Каскадні таблиці стилів) та різноманітні скрипти для створення динаміки на сторінках.

Веб-переглядач клієнтської машини – в більшості випадків це веб-браузер чи веб-аналізатор, що розпочинає «спілкування», подаючи запит на певний ресурс за допомогою протоколу HTTP чи HTTPS на сервер, і сервер відправляє у відповідь вміст цього ресурсу або повертає браузеру повідомлення про помилку з кодом статусу(HTTP status codes - Коди статусу HTTP), якщо не може цього зробити. Ресурс(запитувана клієнтом інформація), як правило, є справжнім файлом у вторинному сховищі сервера, але це не обов'язково і залежить від способу реалізації веб-сервера.

Хоча основна функція полягає в обслуговуванні клієнтів, повна реалізація протоколу HTTP також включає деякі способи отримання контенту від клієнтів. Цей функціонал використовується для надсилання веб-форм, це набір полів на сторінці, які клієнт заповнює та відправляє серверу, включаючи завантаження файлів, сервер обробляє ці дані, при потребі запам’ятовує та дає відповідь.

Майже всі поширені веб-сервери підтримують також і виконання скриптів на серверній машині використовуючи Active Server Pages (ASP - динамічні серверні сторінки) та PHP (HyPertextPreprocessor - гіпертекстовий препроцесор) або на інших мовах програмування, на разі їх дуже багато. Використовуючи клієнтську обробку інформації можна використати JS(JavaScript – TypeScript) мову програмування веб-сторінок, яка може додати зовнішньої краси на сторінку, динаміки, чи полегшити обробку інформації серверу, потрібно буде її перевірити на сервері перед подальшим опрацюванням, виконується лише на клієнтській машині, за допомогою, наприклад, JSON, про це буде в окремому розділі, JS має змогу обмінюватись даними з сервером. Це означає, що поведінка веб-сервера може бути визначена в окремих конфігураційних файлах, а власне програмне забезпечення сервера залишається незмінним. Зазвичай ця функція використовується для динамічного генерування, на стороні клієнта, документів HTML ("на ходу") на відміну від повернення статичних документів, де попередньо згенеровані документи повертаються. Перший використовується в основному для одержання або відображення змінної інформації з баз даних. Останній, як правило, має набагато більшу швидкість і кешується простими засобами, але не має змоги відображати динамічний контент на веб-сторінці.

Веб-сервери (Рисунок 2.1.) часто бувають вбудованими в різні пристрої, наприклад, принтери, маршрутизатори, веб-камери та обслуговують лише локальні мережі, тобто ті мережі в яких самі і перебувають. Потім веб-сервер може бути застосований як частина системи спостереження або адміністрування відповідного пристрою. Зазвичай це означає, що додаткове програмне забезпечення не потрібно встановлювати на клієнтській машині, оскільки

потрібен лише веб-браузер, який користувач може вибрати сам (який зараз входить до переважаючої більшості операційних систем).

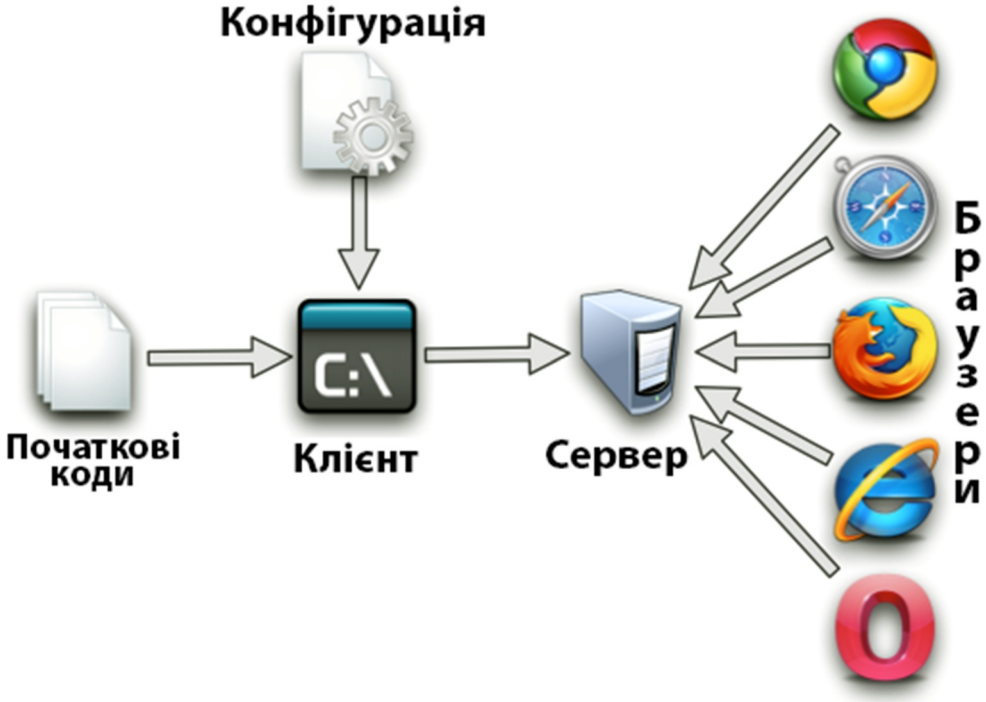


Рисунок 2.1. Веб-сервер.

Веб-сервер (програма) має визначені межі навантаження, оскільки він може обробляти лише певну кількість одночасних клієнтських з'єднань (зазвичай, між 3 і 75 000, за замовчуванням від 400 до 1200 з'єднань) на одну IP-адресу (і один порт TCP), і він може відповідати лише певній максимальній кількісті запитів в секунду (RPS, також відомий як запити в секунду або QPS) залежно від:

- власних налаштувань;
- тип запиту HTTP;
- чи є вміст статичним чи динамічним;
- чи є вміст керованим;
- обмеження програмного забезпечення самого сервера;
- обмеження програмного забезпечення операційною системою комп'ютера, на якій виконується веб-сервер;

Якщо веб-сервер наближається до ліміту або перевищує його межу, він стає таким який більше не відповідає на клієнтські запити, якщо це непередбачено програмним забезпеченням веб-серверу[5].

Причини перевантаження:

1. Перевищений лімітований веб-трафік. Тисячі або навіть сотні тисяч клієнтів, що підключаються до веб-сайту за короткий проміжок часу, для прикладу, ефект «Слешдот» (Slashdot effect);
2. Поширені атаки відмови в обслуговуванні. Атака відмови у наданні обслуговування (DoS-атака – DenialOfService – відмова в обслуговуванні) або розповсюджена атака відмови в обслуговуванні (DDoS-атака - DistributedDenialOfService) - це спроба зробити комп'ютерний або мережевий ресурс недоступним для справжніх користувачів ресурсу;
3. Комп'ютерними черв'яками, які іноді викликають аномально високий трафік через мільйони уражених машин (не пов’язаних між собою);
4. Черви XSS(CrossSiteScripting - Міжсайтовий скриптинг) можуть генерувати надзвичайно високий трафік за допомогою сотень тисяч уражених браузерів клієнтів або інших веб-серверів;
5. Інтернет-трафік ботів не відсортовується і не відрізняється від трафіку справжніх клієнтів або не обмежується на великих веб-сайтах з слабкими ресурсами (мала продуктивність, низька пропускна здатність тощо);
6. Пропускна здатність Інтернету (мережі) знижується так, що запити клієнтів доходять повільніше, а кількість підключень зростає настільки, що досягаються ліміти сервера;
7. Часткова недоступність веб-серверів (машин). Не всі сервери обробляють запити, а тих що обробляють недостатньо. Це може статися через термінові або необхідні технічні обслуговування або оновлення чи зміни в системі, помилки в апаратному чи програмному забезпеченні, збої в роботі резервного копіювання (наприклад, ресурсу чи бази даних) тощо;

Симптоми перевантаження:

- Запити доходять з видимими (можливо, тривалими) затримками (від семи секунд до пари десятків секунд), порівняно з тим як очікується чи як було раніше[7].

- Від веб-сервер повертається код помилки HTTP, такий як 505, 303, 501, 507, 509, 526, 499, 402 або навіть 404, який не підходить під умови перевантаження.
- Веб-сервер відмовляється або розриває (перериває) з'єднання TCP з клієнтом, перш ніж він поверне будь-який вміст.
- У нечастих випадках веб-сервер може повернути лише частину потрібного вмісту для користувача, а потім розриває з'єднання. Таку поведінку можна вважати збоєм, навіть якщо вона зазвичай виникає як симптом перевантаження.

2.2. Обмін даними на основі XML

Поки світ стає свідком безпрецедентної інформаційної революції та великої швидкості зростання потреб застосування баз даних у всіх моментах людського життя. Бази даних напряму пов'язані зі своїм контентом та схемою реалізації, але досі використовують різні частини і структури, які виражають ідентичні поняття та реляції, що може спричинити змістовні та структурні конфлікти.

Потреба генерування XML-даних із різних джерел даних стає все більш значною для широкого спектру середовищ обробки даних. Взаємодія та обмін інформацією між основними системами та іншими системами, що працюють на різних апаратних та програмних платформах, є надзвичайно важливими.

XML(eXtensibleMarkupLanguage) - розширювана мова розмітки, запропонована W3C консорціумом - це спосіб загального опису даних для спрощення обміну інформацією між додатками. Підтримка використання XML важлива для організацій, які потребують обміну інформацією між мейнфреймом з підпрограмами та продуктами, що працюють на різних операційних системах або написані іншими мовами.

За допомогою програмного продукту UltraQuestReporter клієнти можуть генерувати вивід інформації у вигляді XML-документів. У програмі Reporter клієнт може вибрати тип виводу XML, а потім згенерувати вихідний XML на

основі команди CREATE XML NOMAD версії 7.50. За замовчуванням UltraQuest Reporter генерує ієрархічний вихідний XML для імпорту в продукти чи додатки. Прості, без ієрархії, або пласкі файли XML також можуть бути згенеровані для SQL(StructuredQueryLanguage - мова структурованих запитів), орієнтовані для СУБД(СистемаУправлінняБазоюДаних), таких як MySQL, SQL Server або OracleDB.

Схема XML також може бути згенерована для тих додатків, яким потрібна додаткова інформація про структуру та інформацію документу даних XML, який імпортується. Попередню версію структурованого файлу формату XML або DTD(DocumentTypeDefinition - Визначення типу документу) можна створити, але вона буде включена як елемент самого документу даних XML.

У програмах таких як UltraQuest розробники можуть користуватись командою CREATE XML для генерування типів XML-документів, обговорених раніше.

Процедура CREATE XML дозволяє генерувати справжні дійсні документи XML у форматах XML, визначених розробником, використовуючи дані головних комп'ютерів, описаних схемою 4GL. Вихідний XML може зберігатися на головному комп'ютері або передаватися іншим пристроям, де він може використовуватися безпосередньо клієнтськими машинами або перероблятись у формат, який можна застосувати в інших системах. CREATE XML створює чітко визначений, ні від кого незалежний та зрозумілий вихідний XML, який може застосовуватися як і в Java, так і в Visual Basic та у всіх інших додатках з підтримкою XML.

Зазвичай, команда CREATE XML генерує XML-файли, систематизовані одним із двох способів - пласким та ієрархічним. Наступні приклади показують приклади деяких простих файлів XML, створені за допомогою CREATE, щоб показати відмінність між пласкими та ієрархічними даними у XML. У деяких випадках файл XML може бути з поєднанням ієрархічної та пласкої структури.

Плаский XML-документ сильно спрощений порівняно із зовнішнім плоским файлом, наприклад файлом QSAM(QueuedSequentialAccessMethod -

Метод послідовного доступу в чергу). Він має просту структуру XML. Тег <row> додає кожен запис даних. Плаский документ XML напрямлений на імпорт даних у такі додатки, як електронні таблиці або таблиці SQL, які не мають ієрархічної структури. Пласкі документи XML є узагальненими і підходять для використання в різних додатках без необхідності надання конкретних метаданих для супроводження файлу(приклад – Рисунок 2.2.1.).

```
<?xmlversion="1.0"?>
<root>
    <row>
        <state>IF</state>
        <storeName>Veles</storeName>
    </row>
    <row>
        <state>IF</state>
        <storeName>ATB</storeName>
    </row>
    <row>
        <state>IF</state>
        <storeName>Vopak</storeName>
    </row>
    <row>
        <state>IF</state>
        <storeName>VelMart</storeName>
    </row>
</root>
```

Рисунок 2.2.1. Плаский XML-документ згенерований за допомогою CREATE XML.

За замовчуванням CREATE XML генерує ієрархічний файл документа XML. Призначенням ієрархічного XML-файлу є збереження відношень між даними, згенерованих за допомогою CREATE. Ключ сортування представляє батьківське значення, а об'єкти, що сортуються, представляють дітей батьківського ключа. Тег < node – вузол > використовується, щоб вказувати на розрив ієрархічної структури. Теги <вузол> створюються для розривів сортування, спричинених елементами BY (ключами сортування), використаними у запиті. Кожен набір записів у межах розриву сортування вкладається початковим тегом <вузол> та заключним тегом </вузол>. Також варто зазначити

що, одразу після кожного значення сортування теги <вузол> додають кожен окремий дочірній запис, пов’язаний з цим значенням сортування.

Це приклад ієрархічного XML-документа, згенерованого за допомогою CREATE XML. На відміну від плаского XML, значення сортування для ключового стану не повторюються. Значення області IF(Ivano-Frankivsk) з’являється лише один раз для всіх магазинів у Івано-Франківську, а елемент <область> є батьківським елементом <назваМагазину>(Рисунок 2.2.2.).

```
<?xmlversion="1.0"?>
<root>
    <node>
        <state>CN</state>
        <node>
            <storeName>555</storeName>
        </node>
    </node>
    <node>
        <state>IF</state>
        <node>
            <storeName>VelMart</storeName>
        </node>
        <node>
            <storeName>ATB</storeName>
        </node>
        <node>
            <storeName>Vopak</storeName>
        </node>
    </node>
</root>
```

Рисунок 2.2.2. Ієрархічний XML-документ згенерований за допомогою CREATE XML.

2.3. Обмін даними на основі JSON

JSON (JavaScriptObjectNotation - Позначення об'єкта JavaScript) - це простий та зручний у використанні формат обміну інформацією між різними мовами програмування. Люди можуть просто читати і писати, машини легко розбирають код і аналізують. Він заснований на стандарті мови програмування JavaScript 3-го покоління ECMAScript-262(це і є JavaScript). ECMAScript - EuropeanComputer

ManufacturersAssociationScript – Скрипт Європейської Асоціації Виробників Комп'ютерів. JSON - це текстовий формат, який повністю не залежить від мови, але використовує стандарти та правила, знайомі програмістам на мовах сімейства Сі, включаючи C, C++ та C#, а також Ruby, Java, JavaScript, PHP, R, Kotlin, Perl, Python та багато інших. Ці особливості роблять JSON ідеальним засобом для обміну даними між мовами.

JSON побудований на двох конструкціях:

- Колекція пар імен та значень до них. У різних мовах це реалізується по-різному як структура, словник, хеш-таблиця, об'єкт, запис, асоціативний масив чи хеш-масив.
- Впорядкований список значень. У переважаючій більшості мов це реалізується як масив, послідовність, вектор чи список.

Це узагальнені структури даних, які використовуються у всіх мовах програмування, бо це фундаментальна частина програмування. Майже всі сучасні мови програмування підтримують їх у тій чи іншій формі. Варто сказати, що формат даних, який взаємозамінний між мовами програмування, обов’язково повинен базуватися на цих структурах.

Один з найпростіших способів завантаження даних JSON у наші веб-програми - це використання методу AJAX(AsynchronousJavaScriptAndXML - Асинхронний JavaScript і XML), доступного в бібліотеці jQuery(бібліотека JavaScript, призначена для спрощення перетинів та обробки взаємодії із структурою HTML DOM, а також обробки подій, анімацій CSS та AJAX). Легкість отримання даних буде залежати від ресурсу, який надає інформації, але легкий приклад може виглядати так(Рисунок 2.3.).

```
function EXPAJAX(id)
{
    $('tag').html('HTML-code');
    $.ajax({
        type:'GET',
        url:"URL",
        data:"id="+id+"&lang=en-us&format=json&jsoncallback=?",
        success:function(tag) {
            // Зробити щось із відповіддю
        },
        dataType:'jsonp'
    });
}
```

Рисунок 2.3. Отримання даних від ресурсу за допомогою AJAX.

Цей приклад отримує останні дані з того URL, який буде вказано, у форматі JSON та виведе дані у браузер, також він демонструє основи роботи із JSON при зовнішньому джерелі[14].

Бібліотека JSON також використовуються в довідковому прикладі сервера Minnow(дуже маленький і швидкий сервер HTTP(S), який можна легко вбудувати), який використовує Web-Сокети і JSON для обміну інформацією. Простий приклад сервера Minnow показує, як використовувати аналізатор JSON в потоковому режимі, і Web-Сокет вводить дані в пристрій. У прикладі також показано, як використовувати статичний (нединамічний) аналізатор з розпізнавачем JSON та перетворювачем.

2.4. Принцип роботи клієнт–серверної системи

Вирази «клієнт-сервер», «розподілені обрахунки» та «спільна обробка» використовуються вже певний час, але багато в чому є широкими для тлумачення. Насправді принцип, що лежить в обчисленні клієнт-сервер, був основоположною частиною мови програмування COBOL (COmmonBusinessOrientedLanguage - загальна мова, орієнтована на бізнес) з самого зародку, але теперішнє вживання має тенденцію приховувати фактичні функції, які виконуються.

Використовуючи інструменти та послуги компанії Micro Focus(Micro Focus International plc - це британський мультинаціональний бізнес з програмного забезпечення та інформаційних технологій, що базується в Англії), можна реалізувати повне закінчене рішення, яке використовує всю потужність існуючих серверів, використовуючи при цьому обширну встановлену базу настільних клієнтських машин нижнього рівня.

Також слід оглянути можливість написання програми або імплементація до існуючого коду проекту архітектури клієнт-сервер, якщо необхідно виконати один або набір з наступних стратегічних критеріїв:

1. Замінити консольний інтерфейс роботи з застосунком на графічний інтерфейс користувача (GUI - GraphicalUserInterface), наприклад, замінивши 256 екранів у програмі на головній машині на початковому етапі GUI.
2. Проаналізувати вже існуючі правила роботи бізнесу на масштабованих, багатопроцесорних UNIX(назва торгової марки)-подібних системах, налаштованих на багатокористувацький та високошвидкісний доступ до RDBMS(RelationalDataBaseManagementSystem - Реляційна система управління базами даних).
3. Використовувати можливості обробки та управління даними головної машини в дочірніх до головної різних середовищах (наприклад, ПК та UNIX).
4. Контроль за доступом до лімітованого ресурсу.
5. Розподілити обчислювальні навантаження, щоб максимально рівномірно їх розподілити по всіх доступних машинах.
6. Вибирати різні платформи, які забезпечують найбільш придатне рішення для кожної частини програми, а не робити компроміси на одній платформі для всіх частин або переносити всі частини на одну платформу.
7. Повністю використовувати потужність настільних систем персональних комп’ютерів.
8. Застосовувати відповідні дії та аналіз даних локального клієнта.

9. Збільшити рівень доступності програми (наприклад, розробка графічного інтерфейсу для користувачів ПК, а для UNIX – консольна програма, якщо і на цій платформі багато користувачі, то і для них зробити перевагу використання саме вашої програми).
10. Додати нові машини для обробки даних, не збільшуючи навантаження на основну машину.
11. Зменшення адміністративних витрат шляхом централізації адміністративних завдань.

Відповідні переваги та стратегічні критерії залежатимуть від існуючої архітектури бізнес-системи та доступних ресурсів.

Шари додатку, які розташовані на різних машинах з'єднуються через мережу. Продукти середнього рівня програмного забезпечення зазвичай надають ряд послуг, які надають прозорий доступ до ресурсів у мережі[8].

Технології Micro Focus дозволяють маскувати окремі мережеві API (ApplicationProgrammingInterface – інтерфейс прикладного програмування), знайдені у кожного постачальника мережевих послуг, і маскувати API операційної системи для кожного з цих мережевих провайдерів. Це досягається за допомогою єдиного спільного інтерфейсу для всіх протоколів та операційних систем.

Є велике різноманіття платформ, які можна використовувати для впровадження класичного рішення клієнт-сервер. Оскільки однією з головних переваг архітектури клієнт-сервер є можливість використовувати найбільш придатну машину для кожного виду роботи, можна вибрати найбільш придатну платформу для логіки презентації, ділової логіки та логіки доступу до даних[9].

Для клієнтських платформ, які зазвичай містять логіку презентації (і, можливо, невелику частину ділової логіки), вибір, як правило, робиться з Windows, UNIX-подібних систем, MacOS, Android та IOS.

Платформи сервера, які зазвичай містять ділову логіку та елементи доступу до даних, вибираються відповідно до потреб пропускної можливості програми. Залежно від потреб можна обрати платформу низького та середнього рівня або

багатопроцесорні UNIX-подібні системи, або сервери високого класу, такі як комп'ютерні ферми або мейнфрейми.

Більшість платформ можуть функціонувати і як клієнти, так і сервери, але в таблиці перераховані типи платформ, які зазвичай використовуються при впровадженні рішення для клієнта-сервера Micro Focus.

Таблиця 2.4. Типи платформ для Micro Focus.

Платформи	Клієнт	Сервер
Просто Термінал	+	-
DOS	+	-
Windows	+	-
Windows Server	+	+
NetWare	-	+
UNIX	+	+
Mainframe	-	+

Платформа, яка обирається для кожного шару, залежить від конкретних потреб, наприклад, де саме знаходиться зараз існуюча програма; де логіка презентації, де доступ до баз даних та ділова логіка наразі перебувають і де планується їх перемістити, з урахуванням масштабування; доступність платформ на даний момент або заплановані для користувачів; який рівень досвіду програмістів у написанні клієнт-серверного додатку.

Може бути таке, що MIS-відділ(ManagementInformationSystem - інформаційна система управління) компанії або ISV(IndependentSoftwareVendor - незалежний постачальник програмного забезпечення), розробили окремий додаток для персонального комп'ютера, до якого потрібно застосувати архітектуру клієнт-сервер для задоволення потреб своїх користувачів.

Першим кроком може бути модернізація програми, поділивши весь користувальницький інтерфейс на частини для реалізації інтерфейсу або просто для спрощення обслуговування програми.

Наступним кроком може бути відокремлення бізнес-логіки від всієї іншої логіки та розміщення її на високошвидкісному продуктивному сервері UNIX, тим самим скориставшись більшою потужністю обробки та полегшивши розгортання правок або нових версій програми.

Після цього виникає потреба перенести доступ до даних на іншу машину від ділової логіки або додати існуючі дані до корпоративних даних в базі даних; тобто розмістити сервер даних з доступом до СУБД на окремій машині.

Знову ж таки досвідчені організації передбачають ці зміни заздалегідь, тому вони зі старту своїх проектів розробляють та створюють нові додатки на основі своєї гнучкої архітектури клієнт-сервер[16].

В деяких випадках, коли переноситься великий проект, варто не переносити додаток, що витратить більше часу та коштів, а розробити нову програму із масштабованою клієнт-серверною архітектурою.

Модель клієнт-сервер може використовуватися і через мережу глобальну Інтернет, а також лише в локальних мережах (місцевих мережах).

РОЗДІЛ 3. ОПИС ВЗАЄМОДІЇ МІКРОКОНТРОЛЕРА І КЛІЄНТ-СЕРВЕРНОГО КОМПЛЕКСУ

Клієнт-серверний комплекс – розподілена мережева система, в якій виконувані функції розподілені між надавачами послуг - серверами, і отримувачами послуг, іменованими клієнтами, детальніше про це можна прочитати в попередніх розділах.

Проекти та ініціатива від великих гігантів, таких як Планетарна шкіра Cisco, центральна нервова система HP для Землі (CeNSE) та розумний пил, заселять світ мільярдами давачів, підключених до мережі Інтернет. Для досягнення цього завдання нам потрібно вирішити основні виклики, з якими стикаються ці ініціативи та проекти IoT(Інтернету-речей) та WSN. ЕСП8266 - це низько вартісна, високоефективна система-на-кристалі з Wi-Fi на послідовному модулі, частина системи "Розумної платформи підключення Espressif System", яка має на меті забезпечити розробників мобільних платформ інноваційними системами з вбудованими можливостями Wi-Fi як найменшою вартістю та з найбільшою функціональністю[17].

Модуль має різні варіації: ЕСП8266-xx (01-13). Кожен модуль є лише розвитком попереднього модулю з точки зору апаратних можливостей, причому ЕСП8266-01 є найдешевшим, але з мінімальними можливостями, в свою чергу щоб ЕСП8266-13 був найдорожчим з максимальними можливостями. Різні варіації включають різну кількість контактів GPIO, наявність «щита» від перешкод, антени, типу упаковки (крізь отвір або поверхневе кріплення), різку кількість пам'яті та можливість обробки зовнішніх аналогових сигналів.

Сама основна плата, ЕСП8266-01, складається з 2-х контактів GPIO, UART-зв’язку, 32-розрядного процесора з низькою споживаною потужністю та антени на платі. Інші модулі також мають можливості введення ADC, SPI, IIC та більшу кількість GPIO-контактів.

Незважаючи на переваги модуля, в Інтернеті є все ще обмежена опублікована інформація, оскільки ці плати все ще відносно нові. Модуль набув популярності наприкінці 2014 календарного року.

3.1. Програмне забезпечення

Програмування та закидання прошивок відбувалось через - Ардуіно IDE. Ардуіно IDE - це програмне забезпечення з відкритими сирцевими кодами, яке дозволяє легко писати код і завантажувати його у плату Ардуіно. Бібліотека «ЕСП8266 core for Ардуіно» додала підтримку плати ЕСП8266 для інтеграції з Ардуіно IDE. Ардуіно IDE – це багатоплатформове програмне забезпечення, що означає, що воно працює у всіх сучасних операційних системах Windows, Mac OS X або Linux (воно було написане в JAVA). Увесь програмний код міститься у додатках.

Для зв'язку всі ЕСП8266 мають контакти UART - Rx і Tx. Вищі версії, такі як ЕСП8266-12, також мають головний послідовний інтерфейс (SI), який може працювати в двох, трьох або чотирьох провідникових шинах для управління EEPROM або іншими пристроями IIC чи SPI. Насправді, декілька пристроїв IIC з різними адресами пристроїв підтримуються спільним використанням двопровідникової шини. Кілька пристроїв SPI підтримуються спільним використанням сигналів годинника і даних, використовуючи окремі програмні керовані GPIO-контакти в якості вибору мікросхеми. SPI також може використовуватися для управління зовнішніми пристроями, такими як послідовна флеш-пам’ять, аудіо КОДЕКи або іншими підлеглими пристроями.

У ЕСП8266-12 є аналоговий вхід, призначений для обробки аналогових сигналів. Аналогово-цифровий перетворювач (АЦП) має роздільну здатність 10 біт і діапазон напруги від 0 до 1В.

Вищезазначене породжує багато варіантів з точки зору апаратної взаємодії. Поряд із апаратною взаємодією, також легко програмувати ЕСП8266 декількома способами. NodeMCU - це програмний та апаратний набір з відкритим кодом для спрощення проекту Інтернету-речей, використовуючи ЕСП8266 в якості основи та використовуючи інтерактивну мову скриптів LUA для програмування.

3.2. Опис апаратної частини

Апаратна частина складається з плати Ардуіно Нано на базі мікроконтролера Atmega328P та плати ЕСП-01 на базі мікроконтролера ЕСП8266. Плата Ардуіно Нано, детально про плату буде іти мова у наступному розділі, використовується для роботи з існуючим проектом та відповідає за роботу з давачами, всім тим з чого складається проект.

Плата ЕСП-01, детальніше про плату буде розказано у розділі 3.2.2, відповідає за роботу зі стандартом комунікації IEEE 802.11(Wi-Fi), саме він дозволяє реалізувати безпровідне керування.

Тепер детальніше про підключення, живлення на дану схему можна подавати двома способами – за допомогою окремого блока живлення, можна мережевого, можна і батарейним набором на 5В, на контакти плати Ардуіно 5V та GND, а інший спосіб через USB-кабель, так само можна і блоком живлення з відповідним штекером, і є можливість через комп’ютер через USB-гніздо, якщо його вистачить для живлення цілого проекту(і плати Ардуіно і Wi-Fi модулю, і всього іншого що входить у проект). Перемичка на платі Ардуіно між контактами RST та GND слугує для того щоб перевести плату у режим перезавантаження, щоб закинути прошивку у ЕСП-01, схема підключення на Рисунку 3.2.1. При прошиванні контакти Rx і Tx мають бути підключеними попарно, тобто Rx-Rx та Tx-Tx, використовуючи плату Ардуіно в режимі адаптера UART. Відповідно після того як потрібна прошивка буде вже у пам’яті ЕСП8266 перемичку РСТ-ГНД на платі Ардуіно потрібно зняти та поміняти місцями контакти на платі Rx і Tx, щоб між ЕСП та Ардуіно було Rx-Tx та Tx-Rx, тільки в такому випадку ЕСП зможе взаємодіяти з Ардуіно через послідовний порт, схема підключення на Рисунку 3.2.2.

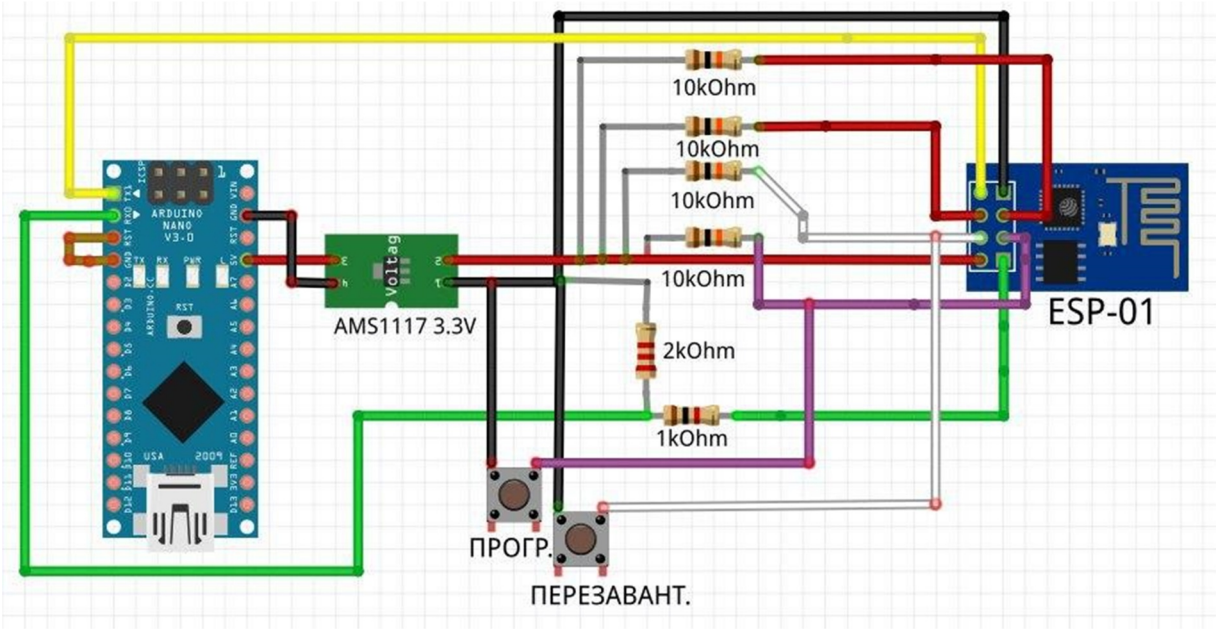


Рисунок 3.2.1. Схема підключення Ардуіно Нано та ЕСП-01 для закидання програмного забезпечення в ЕСП.

Стабілізатор напруги AMS1117 відповідно створює потрібний для живлення модулю ЕСП-01 3.3В перетворюючи з плати Ардуіно 5В. Кнопка програмування відповідає за перехід ЕСП8266 у режим програмування, щоб закинути у флеш-пам’ять прошивку. Кнопка перезавантаження робить перезавантаження плати ЕСП-01.

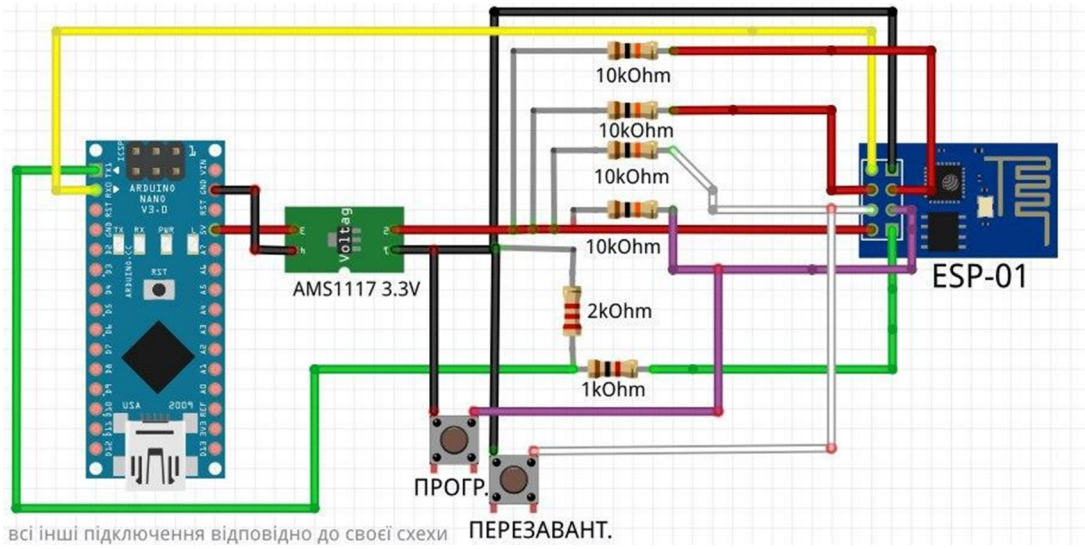


Рисунок 3.2.2. Схема підключення Ардуіно Нано та ЕСП-01 для взаємодії з проектом підключеним до Ардуіно.

Щоб увійти у режим програмування плати ЕСП-01 перед підключенням живлення до проекту треба натиснути кнопку програмування і увімкнути

живлення, і утримувати кнопку до завершення прошивання ЕСП8266. Інший варіант коли вже підключене живлення до проекту, потрібно натиснути і утримувати кнопку програмування, утримуючи цю кнопку натиснути і відпустити кнопку перезавантаження, і далі утримувати кнопку програмування до закінчення закидання прошивки у флеш-пам’ять ЕСП8266.

3.2.1. Ардуіно Нано на базі мікроконтролера Atmega328P

Ардуіно - це маленький комп'ютер, на якому ви можете запрограмувати для читання різноманітну інформацію з навколишнього світу і відправляти свої команди у навколишній світ. Все це можливо тому що ви можете підключити до пристрою Ардуіно кілька пристроїв та компонентів які можна тільки уявляти. Ви можете робити дивовижні проекти з цим комп’ютером, немає обмежень у тому, що ви можете робити та використовувати, все залежить лише від фантазії та навиків.

Ардуіно Нано, це один з видів плат з мікроконтролером, який розроблений молодою командою Ардуіно. Цей мікроконтролер використовує у своїй базі мікроконтролер Atmega168 або Atmega328p. Він трохи схожий на плату Ардуіно Уно, але коли мова доходить до конфігурації контактних площадок та функцій, то Нано плата замінює Уно через невеликі габарити. Відомо, що при проектуванні інтегрованої системи надають перевагу компонентам малого розміру. Різноманіття плат Ардуіно в основному використовується для створення електротехнічних проектів, інтегрованих рішень та для робототехніки і багато іншого. Але плата Нано використовується зазвичай початківцями, які ще не мають значної технічної підготовки.

Плата Нано має багато вбудованих функцій і велику кількість доповнень, наприклад, як плата Ардуіно Duemilanove в порівнянні з платою Уно(це її удосконалена версія). Однак плата Нано відрізняється своєю упаковкою та комплектністю. У ній немає роз'єму для блоку живлення постійного струму, натомість джерело живлення можна подавати за допомогою невеликого міні-USB порту, який напряму підключений до контактів, таких як (+5В)VCC та (мінус

живлення)GND. Ця плата може живитись від 6 до 20 вольт за допомогою міні-порту USB на платі.

Особливості плати Ардуіно Нано в основному включають наступні(Рисунок 3.2.1.а.).

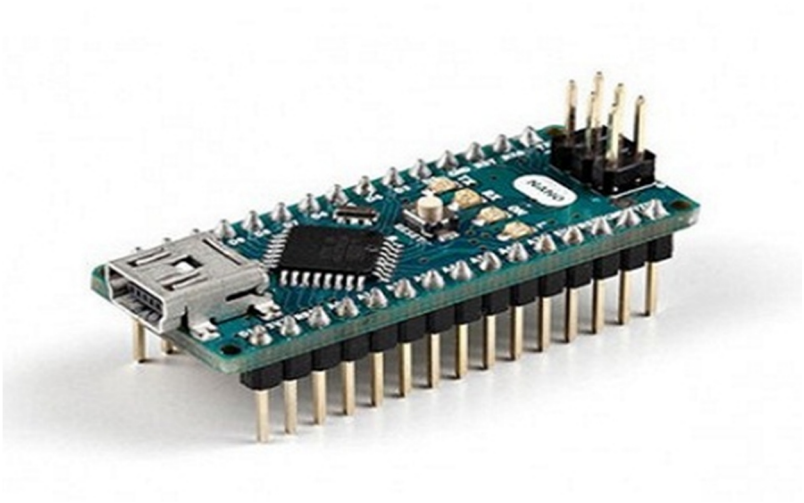


Рисунок 3.2.1.а - Зовнішній вигляд плати Ардуіно Нано.

- Мікроконтролер ATmega328P походить з 8-бітної сімейства AVR;
- Робоча напруга плати +5В;
- Вхідна напруга (Vin) - від 7 до 12 В;
- Кількість контактів вводу / виводу – 22;
- Кількість аналогових контактів вводу / виводу становить 6 від A0 до A5;
- Кількість цифрових контактів – 14;
- Споживання електроенергії плати – від 19 мА;
- Постійний струм на контактах вводу-виводу – 40 мА;
- Флеш-пам’ять - 32 Кб;
- Оперативна пам’ять SRAM - 2 Кб;
- Енергонезалежна пам’ять EEPROM - 1 Кб;
- Частота CLK - 16 МГц;
- Вага приблизно 7г;
- Розмір друкованої плати – 1.8 х 4.5 см;

- Підтримується три види комунікаційних стандартів, такі як SPI, IIC та USART.

Конфігурація контактів Ардуіно Нано показана нижче, і функціональність усіх контактів розглянута нижче(Рисунок 3.2.1.б.).

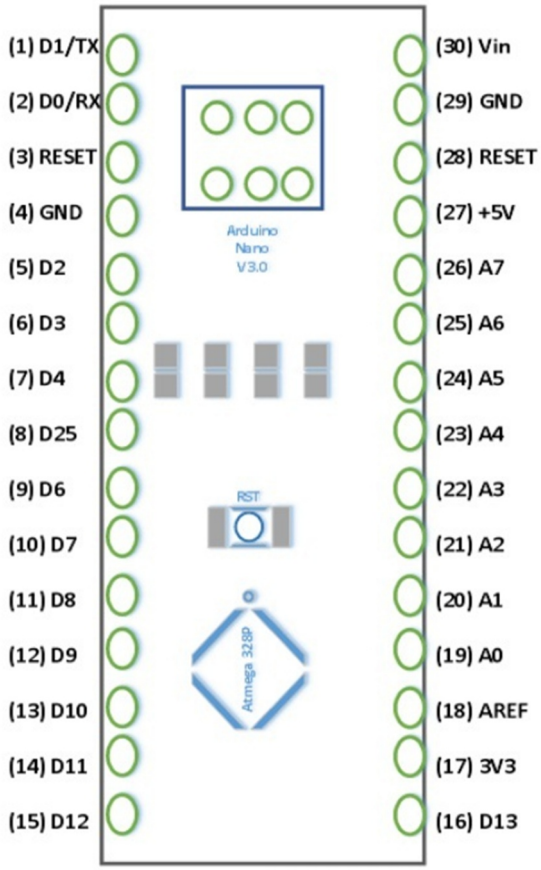


Рисунок 3.2.1.б – Контакти плати Ардуіно Нано.

Контакти живлення, а саме - Vin, 3.3V, 5V та GND:

Vin - це вхідна напруга плати, і вона використовується, коли використовується зовнішній блок живлення від 7В до 12В.

5В - це регульована напруга живлення плати Нано, і вона використовується для подачі живлення на плату, а також на компоненти плати.

3.3В - мінімальна напруга, яка генерується від регулятора напруги на платі.

GND - це заземлений контакт плати.

Ніжка RST(Скидання): Цей контакт використовується для скидання мікроконтролера, перехід плати в режим перезавантаження, наприклад, для заливання прошивки в ЕСП;

Аналогові контакти (A0-A7): ці контакти служать для обчислення аналогової напруги плати в межах від 0 до 5 В

Цифрові контакти вводу / виводу (цифрові контакти від D0 до D13): Ці контакти використовуються як і аналогові контакти, в іншому випадку, з напругою 0V і 5V.

Послідовні контакти (Tx, Rx): ці контакти застосовуються для передачі та отримання послідовних даних TTL.

Зовнішні переривання (2, 3): ці контакти використовуються для активації переривання.

ШІМ, модуляція за часом імпульсу, (3, 5, 6, 9, 11): Ці контакти використовуються для забезпечення 8-бітного виходу ШІМ.

SPI (від 10 по 13 контакт): ці контакти використовуються для підтримки зв'язку SPI, шиною послідовного периферійного інтерфейсу.

Вбудований на платі світлодіод (13): Цей контакт використовується для активації вбудованого на плату світлодіоду.

IIC (A4, A5): Ці контакти використовуються для підтримки TWI-зв'язку.

AREF: Цей контакт використовується для подання опорної напруги на вхідну напругу.

Плата Ардуіно Нано схожа на плату Ардуіно Уно, включаючи подібний мікроконтролер, Atmega328P. Таким чином вони можуть поділитися своїми програмами між собою. Основна відмінність цих двох плат – габаритний розмір. Тому що розмір Ардуіно Уно подвійний у порівнянні з платою Нано. Тож плати Уно використовують більше місця в системі. Програмування Уно можна здійснити за допомогою кабелю USB, тоді як Нано використовує міні-USB-кабель. Основні відмінності між цими двома наведені в таблиці 3.2.1.

Таблиця 3.2.1. Порівняння Ардуіно Уно та Ардуіно Нано.

Специфікації	*Ардуіно Уно*	*Ардуіно Нано*
Процесор	*ATmega328P*	*ATmega328P*
Вхідна напруга	*5В або 7-12В*	*5В або 7-12В*
Частота ЦП	*16 МГц*	*16 МГц*
Аналогові входи\виходи	*6 \ 0*	*8 \ 0*
Цифрові вх і вих\ШІМ	*14 \ 6*	*14 \ 6*
EEPROM \ SRAM, кБ	*1 \ 2*	*1 \ 2*
Флеш-пам'ять, кБ	*32*	*32*
Формат USB	*Звичайний*	*Міні*
USART	*1*	*1*

Передавання інформації використовуючи плату Ардуіно Нано може бути здійснено за допомогою різних комунікацій, таких як використання додаткової плати Ардуіно, зовнішнього комп'ютера, в іншому випадку за допомогою інших мікроконтролерів. Мікроконтролер, що використовує плату Нано (ATmega328), пропонує стандарт послідовного зв'язку (UART TTL). Цей стандарт зв'язку може бути доступний на цифрових контактах плати, таких як TX та RX. Програмне забезпечення Ардуіно складається з послідовного монітора, який дозволяє легко передавати текстову інформацію та отримувати дані від плати.

Світлодіоди TX & RX на платі Нано блиматимуть кожного разу, коли інформація надсилається через посилання FTDI чи USB у напрямку до комп'ютера. Бібліотечний модуль SoftwareSerial надає можливість послідовно передавати інформацію на будь-якому з цифрових контактів на платі. Мікроконтролер також підтримує типи зв'язку SPI & IIC (TWI).

Заливання програмного забезпечення в набір Ардуіно можна здійснити використовуючи програмне забезпечення Ардуіно. Цей компілятор дозволяє заливати нову прошивку не використовуючи зовнішнього устаткування, типу програматора. Також зв'язок може здійснюватися за допомогою протоколу STK500. Також можна уникнути компілятора і програму мікроконтролера можна

виконати використовуючи заголовок послідовного порту в ланцюзі або ICSP з провайдером Ардуіно.

3.2.2. Wi-Fi модуль ЕСП-01 на базі мікроконтролера ЕСП8266

ЕСП8266 - це досить недорогий модуль Wi-Fi. Він надає змогу керувати входами та виходами, як це було би зроблено з Ардуіно, але він працює через Wi-Fi. Враховуючи вище сказане, він чудово підходить для домашньої автоматизації та Інтернету речей.

За допомогою нього можна створювати веб-сервер, надсилати HTTP-запити, керувати виходами, читати входи та переривання, надсилати електронні листи, публікувати твіти, створювати гаджети інтернету речей та багато іншого.

ЕСП-01 - популярна, недорога плата, одна з плат яка використовує мікроконтролер ЕСП8266, але не єдина, є також інші версії з різними параметрами, відрізняються як і пам’яттю, так і кількістю входів-виходів, із вбудованим Wi-Fi.

Мікроконтролери ЕСП8266 призначені для зв'язку через Інтернет через радіосигнали Wi-Fi. Він має вбудовану обробку та невелику за об’ємом пам'ять, що дозволяє інтегрувати його з електронікою через свої GPIO. З ЕСП-01 доступні два контакти GPIO - GPIO0 та GPIO2. (Рисунок 3.2.2.).

Сьогодні мікроконтролери ЕСП8266 були вбудовані у багато плат з мікроконтролерами (наприклад, ЕСП-01, ЕСП-02, ЕСП-12 та інших). Перші мікросхеми ЕСП8266 були розроблені та виготовлені китайською компанією Espressif Systems. Завдяки популярності цих чіпів для них вже є готова збірка всіх мережевих протоколів з підключенням до Wi-Fi, та навіть FTP.

Плата ЕСП-01 має інтегровану на плату антену, радіочастотний блок, підсилювач потужності, фільтри та модуль живлення. ЕСП-01 застосовують в промисловості та багатьох проектах як модуль підключення до Wi-Fi.

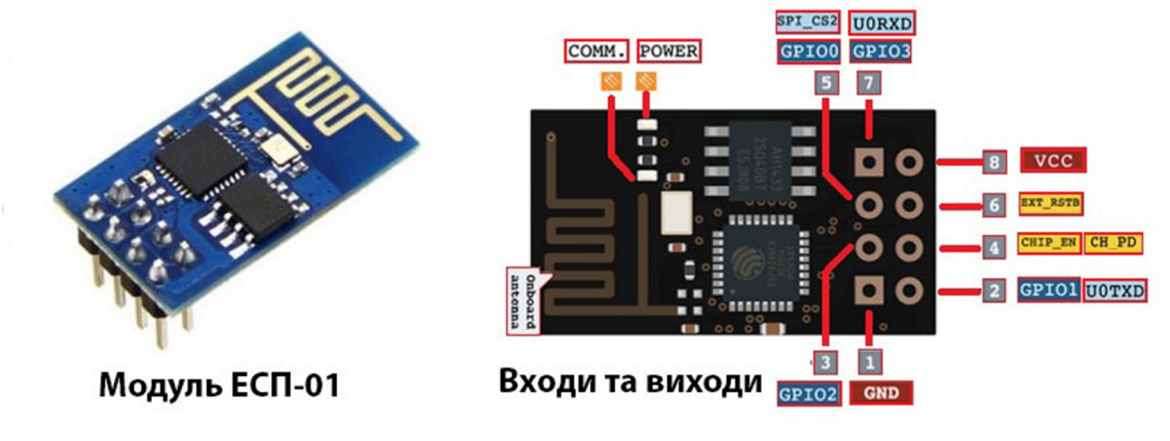


Рисунок 3.2.2. Зовнішній вигляд та входи-виходи ЕСП-01.

Навіть коли мікросхеми працюють на всю «силу», мікросхеми ЕСП8266 вважаються споживачами малої потужності. Мінус може бути в тому, що платі потрібна напруга - 3,3 В постійного струму, на відміну від Ардуіно, де використовується 5В, це може призвести до деяких неприємностей, але питання в різних напругах вирішується використанням у схемі 3.3В стабілізатора по 5В лінії від Ардуіно, струму для живлення ЕСП вистачить. Зазвичай споживання мікросхеми складає 75 - 180 мА. Чіп також підтримує три інших режими живлення: легкий сон (0,48 мА), режим сну модемом (17 мА) і глибокий сон (0,11 мА). У певний час (наприклад, при завантаженні або коли мікросхема "прокидається" з режиму сну) пік робочого струму може досягати до 350 мА.

Потужність постійного струму з напругою 3,3 В подається на модуль ЕСП-01 через контакти на платі живлення VCC та GND. 3.3В також потрібно поставити на контакт CH_PD через резистор 10кОм, щоб не знищити чіп. Кожен контакт вводу / виводу подаватиме струм до 13 мА.

Технічні характеристики ЕСП-01:

В даний час плата ЕСП-01 має дві версії (старша синя версія та новіша чорна версія - також зване ЕСП-01S). Основна відмінність між двома версіями полягає в тому, що новіша чорна версія надає 1 МБ доступної флеш-пам’яті, тоді як синя версія надає лише 512 Кб. Користувачі також виявили, що швидкість передачі даних приблизно на 110-190 bps краще для новішої версії.

- Кількість ядер: 1
- Архітектура: 32-ох розрядна
- Тактова частота процесора: від 80 МГц (зазвичай) – до 160 МГц (програмується)
- Пам'ять: чіп зовнішньої флеш-пам’яті
- Флеш-пам’ять: від 512 Кб - до 1 МБ (залежно від версії)
- Робоча напруга (логічний рівень): 3,6 В постійного струму (максимум)
- Вхідна напруга: 2,8 - 3,6 В постійного струму, найкраще подавати стабілізовані 3,3 В постійного струму
- Джерело живлення: контакти Vcc і GND - обидва потрібно підключити, це відповідно плюс та мінус живлення
- Мережа: Wi-Fi (IEEE 802.11 б / г / н) (недоступна в режимі сну)
- Антена: інтегрована на платі
- Стандарти: FCC і CE, і TELEC та SRRC
- Діапазон частот: 2,4 Гц ~ 2,5 Гц (від 2399 м приблизно до 2502,3 м)
- Підсилення по режимах Wi-Fi: 802,11б: +19 дБм, 802,11г: +16 дБм, 802,11н: +15 дБм
- Чутливість: 802.11б: -89 дБм (12 Мбіт/с), 802.11г: -77 дБм (55 Мбіт/с), 802.11н: -69 дБм (MCS7)
- Режими безпеки: WPA та WPA2
- Типи шифрування: WEP або TKIP, або AES
- Мережеві протоколи: стек протоколів TCP\IP і UDP, і HTTP, і FTP та інші
- Bluetooth: відсутній
- Цифрові контакти вводу / виводу (читання / запис): 2 (GPIO0 та GPIO2)
- Інтегрований на платі світлодіодний контакт: GPIO1 (також подвоюється як Tx контактний)
- Аналогові вхідні контакти: відсутні (керувати можна програмуванням)

- Робочий струм: залежний від режиму, режими сну від 0,11 мА до 17 мА, під час Rx / Tx (Wi-Fi) приблизно 75-180 мА, пік приблизно 350 мА
- Робочий струм на один контакт вводу / виводу: 13 мА
- Розмір плати: 25 х 14,5 мм
- Робоча температура: від -30°С приблизно до 120 °С
- Інтерфейси: послідовний, UART, SDIO, SPI
- Порти: відсутні
- Розмір одного контакту: вилка, 5 х 2,55 мм

Незважаючи на те, що мікросхема ЕСП8266 має 17 ніжок GPIO, проте у ЕСП-01 є лише три робочі контакти (один підключений до синього інтегрованого світлодіоду). Кінцева мета - надати можливість керувати цими трьома контактами за допомогою Wi-Fi. Для того щоб досягти цього, чіп потребує програмного забезпечення - іменованого прошивкою. Прошивка дозволяє отримати доступ до мікросхеми, та визначає, які команди можна використовувати і яким чином на них відповідати. Прошивати ЕСП можна через середовище розробки як і плату Ардуіно, Ардуіно IDE.

Такі терміни, як «заливати» та завантажувати, зазвичай використовуються для процесу додавання або заміни програмного забезпечення мікроконтролера. Програмування (або «закидання» чи «заливання») стосується завантаження (користувальницького або попередньо налаштованого) програмного забезпечення (або мікропрограмного забезпечення) у флеш-пам’ять ЕСП8266 - там, де воно зберігається до його заміни. Закидання здійснюється за допомогою послідовного інтерфейсу UART.

У мікросхемі ЕСП8266 є два різних режими завантаження: "звичайний режим" та "режим програмування". Щоб мати можливість завантажувати нове програмне забезпечення або мікропрограмне забезпечення для свого чіпа, його потрібно завантажувати в режимі програмування. Можна досить легко змінювати режими з одного на інший, досить на контакти GPIO0(ця кнопка увійде у режим програмування) та РСТ(ця кнопка перезавантажить плату) запаяти кнопки на

замикання на ГНД(від’ємний контакт живлення). Закидання замінить попереднє програмне забезпечення (включаючи набір команд AT, які відрізняються між версіями, якщо відповідні було оновлено).

ЕСП8266 має вбудований UART (універсальний асинхронний приймач і передавач), який можна використовувати для послідовного зв'язку TTL та / або заливання прошивки мікросхеми. Щоб мати можливість використовувати UART, потрібен послідовний перетворювач 3,3 В USB в TTL, або використовувати плату Ардуіно як такий перетворювач. Послідовний перетворювач USB в TTL (Рисунок 3.2.2.а.) повинен бути в змозі живити щонайменше 300 мА.

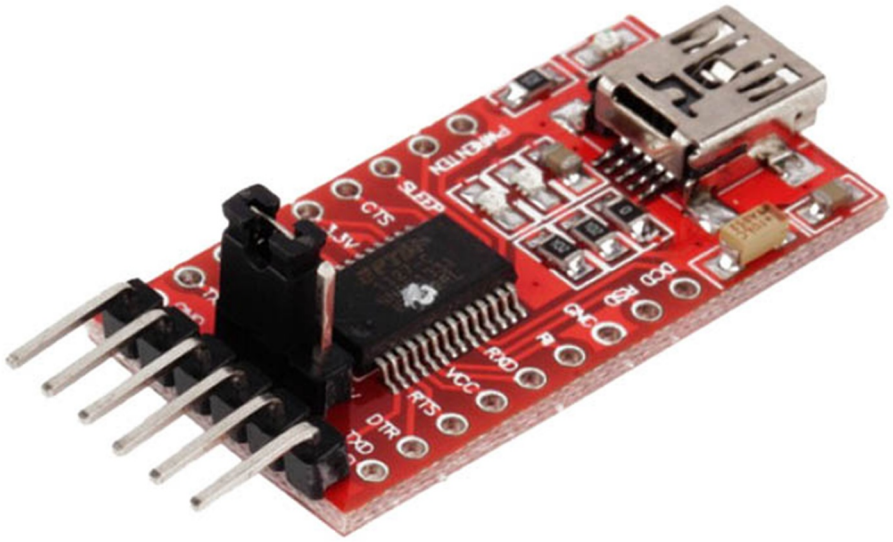


Рисунок 3.2.2.а - Адаптер USB\UART 3.3В.

Доступно кілька варіантів послідовного перетворення на USB. Хороший варіант включає модуль послідовного перетворювача FTDI USB в TTL. Він має перемикач постійного струму 3,3 / 5 В і подаватиме до 500 мА через свій Vcc-контакт живлення на плату ЕСП-01.

Поточну прошивку ЕСП-01 можна замінити за допомогою завантажувального або спеціального програмного забезпечення, написаного відповідним набором для написання та відладки програмного забезпечення (SDK).

Доступно кілька різних наборів програмних засобів, програмного забезпечення та опцій SDK.

Багато ЕСП-01 поставляться заздалегідь запрограмовані з прошивкою, яка використовує команди АТ. Для завантаження в Інтернеті доступні різні варіанти версій програмного забезпечення командного процесора ЕСП8266 АТ.

Коли встановлена відповідна версія програмного забезпечення для процесора команд АТ, команди АТ можна використовувати для зміни деяких технічних налаштувань за замовчуванням або для запуску контактів GPIO. Хоча набір команд АТ зберігається в флеш-пам'яті з цього виходить, що налаштування обладнання, потрібно оновлювати знову після кожного нового завантаження. Команди АТ можуть бути задані за допомогою послідовного інтерфейсу UART або за допомогою зовнішнього програмного забезпечення (програмами для з'єднання через TCP), яке здатне відправляти команди через Wi-Fi за допомогою тієї ж швидкості передачі даних, що була встановлена на ЕСП8266.

За допомогою цієї опції не потрібна жодна конкретна мова програмування, лише команди АТ, щоб мати можливість використовувати модуль ЕСП.

На щастя, компанія Espressif зробила доступним набором для написання та відладки програмного забезпечення (SDK), який дозволив користувачам передавати різні параметри мікропрограмного забезпечення на ЕСП8266.

NodeMCU - одна з найпопулярніших альтернатив для прошивки, що працює на ЕСП8266. Він працює з інтерпретатором Луа на процесорі ЕСП8266, який здатний інтерпретувати команди, написані на мові скриптів Луа (майже як невелика операційна система).

Файли скриптів Луа записуються та зберігаються у форматі .lua. NodeMCU надає змогу користуватись різними GPIO та апаратними функціями, такими як I2C та ШІМ. Ці файли сценаріїв також дозволяють змінювати апаратні настройки для своїх потреб.

За допомогою цієї опції користувач має доступ до широкого спектру функцій Луа та прикладів кодів для написання власних додатків.

Нещодавно спільнотою ЕСП8266 було створено розширення для IDE Ардуіно, яке дозволяє завантажувати програмну прошивку, подібним чином як для Ардуіно (на Сі або С ++). ЕСП8266 додається як плата третьої сторони за допомогою диспетчера плат і робота з нею ведеться як і з Ардуіно.

Цей параметр використовує компільоване рішення. Транслятора немає. Ардуіно IDE бере скетч коду і компілює його, включає відповідні функції бібліотеки і генерує повний окремий фрагмент мікропрограмного забезпечення у вигляді двійкового коду. Потім завантажується весь двійковий код у флеш-пам’ять ЕСП8266.

Файли скриптів (або більш конкретно файли скетчів) записуються та зберігаються у форматі файлів .ino. Скетчі IDE Ардуіно дозволяють функціонувати контактам GPIO та дозволяє змінювати всі апаратні налаштування.

Додаючи ядро ЕСП8266 до Ардуіно IDE, також додається широкий спектр бібліотек та прикладів коду готових рішень, на яких можна навчитись або використати як зразок для написання власного скетчу.

Для досвідчених користувачів існує інструментальна мережа есп-open-sdk, яка дозволяє безпосередньо програмувати на ЕСП8266. Є можливість писати прошивки для ЕСП на Python (MicroPython) та JavaScript (Espruino), про який йшла мова раніше, тобто для написання прошивок існує широкий діапазон налаштувань, додатків та різних мов програмування. Існують різні версії наборів АТ команд від яких залежить підтримка тих, чи інших команд, з повним переліком версій та команд, які входять у певний набір, можна ознайомитись на офіційному сайті розробника ЕСП.

ВИСНОВКИ

1. Як висновок можна сказати що, мета роботи створити веб-застосунок взаємодії мікроконтролера і клієнт-серверного комплексу для віддаленого керування була успішно досягнута та всі завдання були виконані. Технології та інструменти для розробки серверної частини було обрано після вивчення можливих варіантів. Для основних використаних технологій наведено обґрунтування вибору та перелік переваг їх використання у порівнянні з іншими альтернативами.

2. Були проаналізовані різні набори інструментів та визначено необхідний набір для реалізації серверної частини системи. Було виконано розбиття системи на модулі згідно загальноприйнятим правилам побудови архітектури веб-додатків. Для кожного модуля здійснено проектування власної ієрархії та інтерфейсів різного рівня абстракції, визначення функціональних вимог, механізму виконання поставлених перед компонентом задач та його взаємодії з іншими частинами додатку.

3. Був розроблений клієнт-серверний комплекс за допомогою системи-на-кристалі ЕСП8266, яка керує платою Ардуіно Нано на мікроконтролері Атмега, до якої в свою чергу підключений проект, наприклад, розумний електрочайник або розумна розетка чи будь-що інше.

4. Було налаштовано протокол зв’язку між платою Wi-Fi модуля та платою управління за допомогою відправлення команд з ЕСП8266 через послідовний порт у порт Ардуіно, яка була попередньо налаштована ці команди розуміти.

5. Був розроблений зручний та простий у користуванні інтерфейс користувача, з поясненнями, який є HTML веб-сторінкою, яку розуміють всі веб-браузери на всіх пристроях у спільній мережі з ЕСП, вводячи IP-адрес ЕСП у адресний рядок браузера можна бачити інтерфейс.

6. Було проаналізовано роботу усіх компонентів системи і було наведено детальний опис механізмів їх роботи та функціоналу розроблених у межах дії компонентів та інтерфейсів. Використання системи віддаленого керування

значно скорочує час, необхідний для зміни налаштування проекту, що забезпечує зручність використання.

7. Було описано найтиповіші приклади для основних елементів найпопулярніших видів різних клієнт-серверних архітектур.

8. Проаналізовано подальше розширення системи, яке може відбуватись у напрямку збільшення кількості підтримуваних плат, спрощенні розуміння коду, імплементації нового функціоналу, наприклад, за допомогою веб-додатку розробка простого макету з різних елементів, для створення інтерфейсу користувача для віддаленого керування проектом через нього, покращенні програмної оптимізації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ren Chevance. Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions / Ren Chevance –Elsevier/Digital Press, 2005. – 690с.
2. Erich Gamma. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson et al. – Addison-Wesley Professional, 1994. – 458с.
3. Martin Fowler. Patterns of Enterprise Application Architecture / Martin Fowler. – Addison-Wesley Professional, 2002. – 649с.
4. Len Bass. Software Architecture in Practice / Len Bass, Paul Clements, Rick Kazman. – Addison-Wesley Professional, 2012. – 334с.
5. Гонсалвес Э. Изучаем Java EE 7 / Э. Гонсалвес. – Appress, 2016. – 673с.
6. Peter Herzum. Business Component Factory : A Comprehensive Overview of Component-Based Development for the Enterprise / Peter Herzum, Oliver Sims. - Wiley, 1999 – 257с.
7. Jose Sandoval. RESTful Java Web Services / Jose Sandovel. – PACT publishing, 2009. – 328с.
8. Layered Application, Microsoft Developers Network – Режим доступу: https://msdn.microsoft.com/en-us/library/ff650258.aspx - Дата тоступу: 24.05.2020
9. A multi-tier architecture for building RESTful Web services – Режим доступу: http://www.ibm.com/developerworks/library/wa-aj-multitier/ - Дата доступу 27.05.2020
10. Jim Webber. REST in Practice, Hypermedia and Systems Architecture / Jim Webber, Savas Parastatidis,Ian Robbinson. – O`Reilly Media, 2010 – 287с.

11. Головенкін В.П. Рекомендації щодо розробки навчальних планів / В. П. Головенкін. – К. : НТУУ «КПІ», 2012. – 23 с. – 250 прим.
12. Головенкін В.П. Рекомендації щодо розробки навчальних та робочих навчальних планів за новими напрямами підготовки бакалаврів / В. П. Головенкін, А. Д. Лемешко – К.: ІВЦ “Видавництво «Політехніка»”, 2007. – 24 с.
13. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів / Уклад. В.Є. Богданюк, К.В. Березовський, В.П. Пашин та ін. - К.: НТУУ "КПІ", 1999. - 66 с.
14. Обзор Gson - работаем с JSON в Java [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://www.javenue.info/post/gson-json-api. -19.
15. Apache POI [Електронний ресурс] – Режим доступу до ресурсу: https://poi.apache.org/
16. JPA 2 Annotations [Електронний ресурс] – Режим доступу до ресурсу: http://www.objectdb.com/api/java/jpa/annotations.
17. Servlet API [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/servlet-api.
18. Three-Tier Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/24649/three-tier-architecture.

Скетч для ЕСП8266:

```
#include <ESP8266WiFi.h>
// Дані для мережі вай-фай ім'я та пароль
const char* ssid     = "Назва вай-фай точки доступу";
const char* password = "Пароль від вай-фай точки доступу";
// Встановити порт веб-сервера на 80
WiFiServer server(80);
// Змінна щоб зберігати HTTP запит
String header;
// Різні змінні для збереження стану
String output5State = "0"; // змінна для стану режиму
String starState = "1"; // змінна для стану ввімк-вимк
String okState = "0"; // змінна для загальних налаштувань
String table = "<table>\
        <tbody>\
            <tr>\
                <th>Номер режиму</th>\
                <th>Режим</th>\
                <th>Кнопки ← та →</th>\
                <th>Кнопки ↑ та ↓</th>\
                <th>Кнопка підрежиму</th>\
            </tr>\
            <tr>\
                <td>1</td>\
                <td>Шкала гучності (градієнт)</td>\
                <td>Плавність анімації</td>\
                <td>---</td>\
                <td>---</td>\
            </tr>\
            <tr>\
                <td>2</td>\
                <td>Шкала гучності (веселка)</td>\
                <td>Плавність анімації</td>\
                <td>Швидкість веселки</td>\
                <td>---</td>\
            </tr>\
            <tr>\
                <td>3</td>\
                <td>Світломузика (5 смуг)</td>\
                <td>Плавність анімації</td>\
                <td>Чутливість</td>\
                <td>---</td>\
            </tr>\
            <tr>\
                <td>4</td>\
                <td>Світломузика (3 смуги)</td>\
                <td>Плавність анімації</td>\
                <td>Чутливість</td>\
                <td>---</td>\
```

```
</tr>\
<tr>\
    <td>5</td>\
    <td>Світломузика (1 смуга)</td>\
    <td>---</td>\
    <td>---</td>\
    <td>---</td>\
</tr>\
<tr>\
    <td>5,1 підрежим</td>\
    <td>3 Частоти</td>\
    <td>Плавність анімації</td>\
    <td>Чутливість</td>\
    <td>Зміна підрежиму</td>\
</tr>\
<tr>\
    <td>5,2 підрежим</td>\
    <td>Низькі</td>\
    <td>Плавність анімації</td>\
    <td>Чутливість</td>\
    <td>Зміна підрежиму</td>\
</tr>\
<tr>\
    <td>5,3 підрежим</td>\
    <td>Середні</td>\
    <td>Плавність анімації</td>\
    <td>Чутливість</td>\
    <td>Зміна підрежиму</td>\
</tr>\
<tr>\
    <td>5,4 підрежим</td>\
    <td>Високі</td>\
    <td>Плавність анімації</td>\
    <td>Чутливість</td>\
    <td>Зміна підрежиму</td>\
</tr>\
<tr>\
    <td>6</td>\
    <td>Стробоскоп</td>\
    <td>Плавність спалахів</td>\
    <td>Частота спалахів</td>\
    <td>---</td>\
</tr>\
<tr>\
    <td>7</td>\
    <td>Кольорове підсвічування</td>\
    <td>---</td>\
    <td>---</td>\
    <td>---</td>\
</tr>\
<tr>\
```

```
        <td>7,1 підрежим</td>\
        <td>Постійний</td>\
        <td>Колір</td>\
        <td>Насиченість</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>7,2 підрежим</td>\
        <td>Плавна зміна кольору</td>\
        <td>Швидкість</td>\
        <td>Насиченість</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>7,3 підрежим</td>\
        <td>Біжуча Веселка</td>\
        <td>Швидкість</td>\
        <td>Крок веселки</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>8</td>\
        <td>Біжучі Частоти</td>\
        <td>---</td>\
        <td>---</td>\
        <td>---</td>\
    </tr>\
    <tr>\
        <td>8,1 підрежим</td>\
        <td>3 Частоти</td>\
        <td>Швидкість</td>\
        <td>Чутливість</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>8,2 підрежим</td>\
        <td>Низькі</td>\
        <td>Швидкість</td>\
        <td>Чутливість</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>8,3 підрежим</td>\
        <td>Середні</td>\
        <td>Швидкість</td>\
        <td>Чутливість</td>\
        <td>Зміна підрежиму</td>\
    </tr>\
    <tr>\
        <td>8,4 підрежим</td>\
        <td>Високі</td>\
```

```
                <td>Швидкість</td>\
                <td>Чутливість</td>\
                <td>Зміна підрежиму</td>\
            </tr>\
            <tr>\
                <td>9</td>\
                <td>Аналізатор спектру</td>\
                <td>Крок кольору</td>\
                <td>Колір</td>\
                <td>---</td>\
            </tr>\
            <tr>\
                <td rowspan='2'>Загальні Налаштування (перемикач)</td>\
                <td rowspan='2'>Всі режими</td>\
                <td rowspan='2'>Яскравість палаючих світлодіодів</td>\
                <td rowspan='2'>Яскравість 'не палаючих' світлодіодів</td>\
                <td>---</td>\
            </tr>\
            <tr>\
                <td></td>\
            </tr>\
            <tr>\
                <td colspan='5'>Інші кнопки: Калібрування шумів, * – ввімк/вимк
системи.</td>\
            </tr>\
        </tbody>\
    </table>";
void setup() {
  Serial.begin(9600); //швидкість 9600 як і в Ардуіно
  // Підключення до мережі Wi-Fi за допомогою SSID та пароля
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Прінт локального IP-адресу та запуск веб-сервера
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
void loop(){
  WiFiClient client = server.available();   // Слухати клієнтів
  if (client) {                             // Якщо підключиться новий клієнт,
    // Serial.println("New Client.");       // роздрукувати повідомлення в
послідовний порт
    String currentLine = "";                // зробити String для зберігання
вхідних даних від клієнта
```

```
    while (client.connected()) {            // цикл, поки клієнт підключений
      if (client.available()) {             // якщо з клієнта є байти,
        char c = client.read();             // тоді читати байти, потім
        // Serial.write(c);                 // роздрукувати їх на послідовному
моніторі
        header += c;
        if (c == '\n') {                    // якщо байт - це символ нового рядка
          // якщо поточний рядок порожній, ви отримали два символи нового рядка
поспіль.
          // ось завершення HTTP-запиту клієнта, тому надішліть відповідь:
          if (currentLine.length() == 0) {
            // Заголовки HTTP завжди починаються з коду відповіді (наприклад,
HTTP/1.1 200 OK)
            // і тип вмісту, щоб клієнт знав, що відбувається, а потім порожній
рядок:
             client.println("HTTP/1.1 200 OK");
             client.println("Content-type:text/html");
             client.println("Connection: close");
            client.println();
            // Вивід в послідовний порт тексту з нажатою кнопкою
            // щоб ардуіно зловила це і зробила те що просять
            if (header.indexOf("GET /1/on") >= 0) {
              Serial.println("button_1");
              output5State = "1";
            } else if (header.indexOf("GET /2/on") >= 0) {
              Serial.println("button_2");
              output5State = "2";
            } else if (header.indexOf("GET /3/on") >= 0) {
              Serial.println("button_3");
              output5State = "3";
            } else if (header.indexOf("GET /4/on") >= 0) {
              Serial.println("button_4");
              output5State = "4";
            }else if (header.indexOf("GET /5/on") >= 0) {
              Serial.println("button_5");
              output5State = "5";
            }else if (header.indexOf("GET /6/on") >= 0) {
              Serial.println("button_6");
              output5State = "6";
            }else if (header.indexOf("GET /7/on") >= 0) {
              Serial.println("button_7");
              output5State = "7";
            }else if (header.indexOf("GET /8/on") >= 0) {
              Serial.println("button_8");
              output5State = "8";
            }else if (header.indexOf("GET /9/on") >= 0) {
              Serial.println("button_9");
              output5State = "9";
            }  else if (header.indexOf("GET /0/on") >= 0) {
              Serial.println("button_0");
            }  else if (header.indexOf("GET /star/on") >= 0) {
```

```
            Serial.println("button_star");
            starState = "1";
          }   else if (header.indexOf("GET /star/off") >= 0) {
            Serial.println("button_star");
            starState = "0";
          }  else if (header.indexOf("GET /hash") >= 0) {
            Serial.println("button_hash");
          } else if (header.indexOf("GET /ok/on") >= 0) {
            Serial.println("button_ok");
            okState = "1";
          } else if (header.indexOf("GET /ok/off") >= 0) {
            Serial.println("button_ok");
            okState = "0";
          } else if (header.indexOf("GET /left") >= 0) {
            Serial.println("button_left");
          } else if (header.indexOf("GET /right") >= 0) {
            Serial.println("button_right");
          } else if (header.indexOf("GET /up") >= 0) {
            Serial.println("button_up");
          } else if (header.indexOf("GET /down") >= 0) {
            Serial.println("button_down");
          }
          // Показ HTML сторінки
          client.println("<!DOCTYPE html><html lang=\"uk\">");
          client.println("<head><meta charset=\"UTF-8\"><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
          client.println("<link rel=\"icon\" href=\"data:,\"><title>ColorMusic
Page Control</title>");
          // CSS для стилізації кнопок включення / вимкнення
          // Не соромтеся змінювати атрибути кольору тла та розміру шрифту
відповідно до ваших уподобань
          client.println("<style>");
          client.println(".button { background-color: #56d54c; border: none;
color: white; padding: 16px 40px;");
          client.println("text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}");
          client.println("html, body, section{height: 100%;} body {color:
#fff;font-family: sans-serif;font-size: 1.25rem;line-height: 150%;text-align:
center;}div {display: flex;flex-direction: column;justify-content: center;}");
          client.println("h1{font-size: 1.75rem;margin: 0 0 0.75rem
0;}.container{display: flex;}.left-half {background-color:#b7b7b7;flex: 1;padding:
1rem;}.right-half {background-color:#79a1f1;flex: 1;padding: 1rem;}"); // сторінка
флексами ділиться навпіл
          client.println("table{border-collapse: collapse;width: 100%;font-size:
0.75rem;line-height: 90%;}th,td{padding: 7px;text-align: center;border-bottom: 1px
solid #ddd; font-weight: bold;}th{color:#303030;}"); //стилі таблиці
          client.println(".button2 {background-color:
#d7b050;}</style></head>");
          // частина боді веб-сторінки
          client.println("<body>");
          client.println("<section class='container'>");
```

```
            // Початок лівої частини
            client.println("<div class='left-half'><article><h2>Пояснення
керування</h2>");
            client.println(table);
            client.println("</article></div>");
            // КІНЕЦЬ лівої частини
            // Початок правої частини
            client.println("<div class='right-half'><article><h2>КЕРУВАННЯ</h2>");
            // Початок правої частини
            if (output5State=="1") {
              client.println("<p><a href=\"/2/on\"><button class=\"button\">1
РЕЖИМ</button></a></p>");
            } else if (output5State=="2"){
              client.println("<p><a href=\"/3/on\"><button class=\"button
button2\">2 РЕЖИМ</button></a></p>");
            } else if (output5State=="3") {
              client.println("<p><a href=\"/4/on\"><button class=\"button\">3
РЕЖИМ</button></a></p>");
            } else if (output5State=="4"){
              client.println("<p><a href=\"/5/on\"><button class=\"button
button2\">4 РЕЖИМ</button></a></p>");
            } else if (output5State=="5") {
              client.println("<p><a href=\"/6/on\"><button class=\"button\">5
РЕЖИМ</button></a></p>");
            } else if (output5State=="6"){
              client.println("<p><a href=\"/7/on\"><button class=\"button
button2\">6 РЕЖИМ</button></a></p>");
            } else if (output5State=="7") {
              client.println("<p><a href=\"/8/on\"><button class=\"button\">7
РЕЖИМ</button></a></p>");
            }else if (output5State=="8") {
              client.println("<p><a href=\"/9/on\"><button class=\"button
button2\">8 РЕЖИМ</button></a></p>");
            }else if (output5State=="9") {
              client.println("<p><a href=\"/1/on\"><button class=\"button\">9
РЕЖИМ</button></a></p>");
            }
            else {
                client.println("<p><a href=\"/1/on\"><button
class=\"button\">Натисніть, щоб встановити 1 режим!</button></a></p>");
            }
            // кнопка калібровки
            client.println("<p><a href=\"/0/on\"><button
class=\"button\">Калібрування шумів!</button></a></p>");
            //кнопка ВКЛ і ВИКЛ
            if (starState=="1") {
              client.println("<p><a href=\"/star/off\"><button
class=\"button\">Систему Ввімкнено!</button></a></p>");
            } else {
              client.println("<p><a href=\"/star/on\"><button class=\"button
button2\">Систему Вимкнено!</button></a></p>");
```

```
            }
            // копка зміни підрежиму
            client.println("<p><a href=\"/hash\"><button class=\"button\">Змінити
підрежим!</button></a></p>");
            //кнопка ОК Загальних налаштувань
            if (okState=="0") {
              client.println("<p><a href=\"/ok/on\"><button class=\"button
button2\">Загальні налаштування Вимкнено!</button></a></p>");
            } else {
              client.println("<p><a href=\"/ok/off\"><button
class=\"button\">Загальні налаштування Ввімкнено!</button></a></p>");
            }
            // кнопка вверх-вниз
            client.println("<p><a href=\"/up\"><button
class=\"button\">↑↑↑</button></a></p>");
            client.println("<p><a href=\"/down\"><button
class=\"button\">↓↓↓</button></a></p>");
            // кнопка вправо вліво
            client.println("<p><a href=\"/left\"><button
class=\"button\">←←←</button></a></p>");
            client.println("<p><a href=\"/right\"><button
class=\"button\">→→→</button></a></p>");
            // Кінець правої частини
            client.println("</article></div>");
            // Кінець правої частини
            client.println("</section>");
            client.println("</body></html>");
            // Відповідь HTTP закінчується ще одним порожнім рядком
            client.println();
            // Вирвіть з циклу вайл
            break;
          } else { // якщо у вас новий рядок, очистіть поточний рядок
            currentLine = "";
          }} else if (c != '\r') {  // якщо у вас є щось інше, крім символу
повернення символу,
          currentLine += c;      // додайте його до кінця поточного рядка
        }}}
    // Очистіть змінну header
    header = "";
    // Закрити з'єднання
    client.stop();
    // Serial.println("Client disconnected.");
    // Serial.println("");
  }}
```

Скетч для Ардуіно:

```
// ************************** НАЛАШТУВАННЯ **************************
// ----- налаштування дистанційного керування
#define REMOTE_TYPE 1      // 0 - без дистанційного керування, 1 - веб через ЕСП
// ----- налаштування параметрів
#define KEEP_SETTINGS 1     // зберігати ВСі налаштування в енергонезалежній пам'яті
#define KEEP_STATE 1               // зберігати в пам'яті стан вкл / викл системи (через дистанційне керування)
#define RESET_SETTINGS 0    // скидання налаштувань в EEPROM пам'яті (поставити 1, прошитись, поставити назад 0, прошитись. і все)
#define SETTINGS_LOG 0      // виведення всіх налаштувань з EEPROM в порт при запуску
// ----- налаштування стрічки
#define NUM_LEDS 120        // кількість світлодіодів (дана версія підтримує до 410 штук)
#define CURRENT_LIMIT 3000  // ліміт по струму в міліамперах, автоматично керує яскравістю (пам'ятай про свій блок живлення!), 0 - вимкнути ліміт
byte BRIGHTNESS = 220;      // яскравість за замовчуванням (0 - 255)
// ----- контакти підключення
#define SOUND_R A2          // аналоговий пін вхід аудіо, правий канал
#define SOUND_L A1          // аналоговий пін вхід аудіо, лівий канал
#define SOUND_R_FREQ A3     // аналоговий пін вхід аудіо для режиму з частотами (через конденсатор)
#define BTN_PIN 3           // кнопка перемикання режимів (PIN --- КНОПКА --- GND)
#if defined(__AVR_ATmega32U4__) // Піни для Arduino Pro Micro
#define MLED_PIN 17             // пін світлодіода режимів на ProMicro
#define MLED_ON LOW
#define LED_PIN 9               // пін DI світлодіодної стрічки на ProMicro
#else                           // Піни для інших плат Arduino (за замовчуванням)
#define MLED_PIN 13             // пін світлодіода режимів
#define MLED_ON HIGH
#define LED_PIN 12              // пін DI світлодіодної стрічки
#endif
#define POT_GND A0              // пін землі для потенціометра
#define IR_PIN 2                // пін ІЧ приймача
// ----- настройки веселки
float RAINBOW_STEP = 5.00;         // крок зміни кольору веселки
// ----- відтворення
#define MODE 0                    // режим при запуску
#define MAIN_LOOP 5               // період основного циклу відтворення (за замовчуванням 5)
// ----- сигнал
#define MONO 1                    // 1 - тільки один канал (ПРАВИЙ!!!!! SOUND_R !!!!!), 0 - два канали
#define EXP 1.4                   // ступінь посилення сигналу (для більш "різкої" роботи) (за замовчуванням 1.4)
#define POTENT 1                  // 1 - використовуємо потенціометр, 0 - використовується внутрішнє джерело опорної напруги 1.1 В
```

```
byte EMPTY_BRIGHT = 30;           // яскравість "не палаючих" світлодіодів (0 - 255)
#define EMPTY_COLOR HUE_PURPLE    // колір "не палаючих" світлодіодів. Буде чорний, якщо яскравість 0
// ----- нижній поріг шумів
uint16_t LOW_PASS = 100;          // нижній поріг шумів режим VU, ручна настройка
uint16_t SPEKTR_LOW_PASS = 40;    // нижній поріг шумів режим спектра, ручна настройка
#define AUTO_LOW_PASS 1           // дозволити настройку нижнього порога шумів при запуску (стандартно 0)
#define EEPROM_LOW_PASS 0         // поріг шумів зберігається в енергонезалежній пам'яті (стандартно 1)
#define LOW_PASS_ADD 13           // "додаткова" величина до нижнього порогу, для надійності (режим VU)
#define LOW_PASS_FREQ_ADD 3       // "додаткова" величина до нижнього порогу, для надійності (режим частот)
// ----- режим шкала гучності
float SMOOTH = 0.3;               // коефіцієнт плавності анімації VU (за замовчуванням 0.5)
#define MAX_COEF 1.8              // коефіцієнт гучності (максимальне дорівнює середньому * на цей коефіцієнт) (за замовчуванням 1.8)
// ----- режим світломузики
float SMOOTH_FREQ = 0.8;          // коефіцієнт плавності анімації частот (за замовчуванням 0.8)
float MAX_COEF_FREQ = 1.2;        // коефіцієнт порогу для "спалаху" світломузики (за замовчуванням 1.5)
#define SMOOTH_STEP 20            // крок зменшення яскравості в режимі світломузики (чим більше, тим швидше гасне)
#define LOW_COLOR HUE_RED         // колір низьких частот
#define MID_COLOR HUE_GREEN       // колір середніх частот
#define HIGH_COLOR HUE_YELLOW     // колір високих частот
// ----- режим стробоскопа
uint16_t STROBE_PERIOD = 140;     // період спалахів, мілісекунди
#define STROBE_DUTY 20            // шпаруватість спалахів (1 - 99) - відношення часу спалаху до часу темряви
#define STROBE_COLOR HUE_YELLOW   // колір стробоскопа
#define STROBE_SAT 10              // насиченість. Якщо 0 - колір буде БІЛИЙ при будь-якому кольорі (0 - 255)
byte STROBE_SMOOTH = 200;         // швидкість наростання / згасання спалаху (0 - 255)
// ----- режим підсвічування
byte LIGHT_COLOR = 0;             // початковий колір підсвічування
byte LIGHT_SAT = 255;             // початкова насиченість підсвічування
byte COLOR_SPEED = 100;
int RAINBOW_PERIOD = 1;
float RAINBOW_STEP_2 = 0.5;
// ----- режим біжучих частот
byte RUNNING_SPEED = 11; // швидкість
// ----- режим аналізатора спектра
byte HUE_START = 0;
byte HUE_STEP = 5;
```

```
#define LIGHT_SMOOTH 2
/*
  кольори для HSV
  HUE_RED
  HUE_ORANGE
  HUE_YELLOW
  HUE_GREEN
  HUE_AQUA
  HUE_BLUE
  HUE_PURPLE
  HUE_PINK
*/
// ----- КНОПКИ веб-ПУЛЬТА -----
#if REMOTE_TYPE == 1
String BUTT_UP    =  "button_up\r\n";
String BUTT_DOWN  = "button_down\r\n";
String BUTT_LEFT  = "button_left\r\n";
String BUTT_RIGHT = "button_right\r\n";
String BUTT_OK    = "button_ok\r\n";
String BUTT_1     = "button_1\r\n";
String BUTT_2     = "button_2\r\n";
String BUTT_3     = "button_3\r\n";
String BUTT_4     = "button_4\r\n";
String BUTT_5     = "button_5\r\n";
String BUTT_6     = "button_6\r\n";
String BUTT_7     = "button_7\r\n";
String BUTT_8     = "button_8\r\n";
String BUTT_9     = "button_9\r\n";
String BUTT_0     = "button_0\r\n";
String BUTT_STAR  = "button_star\r\n";
String BUTT_HASH  = "button_hash\r\n";
#endif
// ------------------------------ ДЛЯ РОЗРОБНИКІВ ------------------------------
#define MODE_AMOUNT 9      // кількість режимів
#define STRIPE NUM_LEDS / 5
float freq_to_stripe = NUM_LEDS / 40; // /2 так як симетрія, і / 20 так як 20
частот
#define FHT_N 64          // ширина спектру х2
#define LOG_OUT 1
#include <FHT.h>          // перетворення Хартлі
#include <EEPROMex.h>
#define FASTLED_ALLOW_INTERRUPTS 1
#include "FastLED.h"
CRGB leds[NUM_LEDS];
#include "GyverButton.h"
GButton butt1(BTN_PIN);
#include "IRLremote.h"
CHashIR IRLremote;
String IRdata;
// градієнт-палітра від зеленого до червоного
DEFINE_GRADIENT_PALETTE(soundlevel_gp) {
```

```
  0,    0,    255,  0,  // зелений
  100,  255,  255,  0,  // жовтий
  150,  255,  100,  0,  // помаранчевий
  200,  255,  50,   0,  // червоний
  255,  255,  0,    0   // червоніший
};
CRGBPalette32 myPal = soundlevel_gp;
int Rlenght, Llenght;
float RsoundLevel, RsoundLevel_f;
float LsoundLevel, LsoundLevel_f;
float averageLevel = 50;
int maxLevel = 100;
int MAX_CH = NUM_LEDS / 2;
int hue;
unsigned long main_timer, hue_timer, strobe_timer, running_timer, color_timer,
rainbow_timer, eeprom_timer;
float averK = 0.006;
byte count;
float index = (float)255 / MAX_CH;   // коефіцієнт переведення для палітри
boolean lowFlag;
byte low_pass;
int RcurrentLevel, LcurrentLevel;
int colorMusic[3];
float colorMusic_f[3], colorMusic_aver[3];
boolean colorMusicFlash[3], strobeUp_flag, strobeDwn_flag;
byte this_mode = MODE;
int thisBright[3], strobe_bright = 0;
unsigned int light_time = STROBE_PERIOD * STROBE_DUTY / 100;
volatile boolean ir_flag;
boolean settings_mode, ONstate = true;
int8_t freq_strobe_mode, light_mode;
int freq_max;
float freq_max_f, rainbow_steps;
int freq_f[32];
int this_color;
boolean running_flag[3], eeprom_flag;
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
// ------------------------------ ДЛЯ РОЗРОБНИКІВ ------------------------------
void setup() {
  Serial.begin(9600);
  FastLED.addLeds<WS2811, LED_PIN, GRB>(leds, NUM_LEDS).setCorrection(
TypicalLEDStrip );
  if (CURRENT_LIMIT > 0) FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT_LIMIT);
  FastLED.setBrightness(BRIGHTNESS);
#if defined(__AVR_ATmega32U4__)    //Вимкнення світлодіодів на Pro Micro
  TXLED1;                           //на ProMicro вимкнемо і TXLED
  delay (1000);                     //При живленні по usb від комп'ютера потрібна
затримка перед вимиканням RXLED. Якщо живити від БП, то можна прибрати цей рядок.
#endif
  pinMode(MLED_PIN, OUTPUT);        //Режим піна для світлодіода режиму на вихід
```

```
  digitalWrite(MLED_PIN, !MLED_ON); //Вимкнення режиму світлодіода
  pinMode(POT_GND, OUTPUT);
  digitalWrite(POT_GND, LOW);
  butt1.setTimeout(900);
  IRLremote.begin(IR_PIN);
  // для збільшення точності зменшуємо опорну напругу,
  // виставивши EXTERNAL і підключивши Aref до виходу 3.3V на платі через дільник
  // GND ---[10-20 кОм] --- REF --- [10 кОм] --- 3V3
  // в даній схемі GND береться з A0 для зручності підключення
  if (POTENT) analogReference(EXTERNAL);
  else
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
    analogReference(INTERNAL1V1);
#else
    analogReference(INTERNAL);
#endif
  sbi(ADCSRA, ADPS2);
  cbi(ADCSRA, ADPS1);
  sbi(ADCSRA, ADPS0);
  if (RESET_SETTINGS) EEPROM.write(100, 0);        // скидання прапора налаштувань
  if (AUTO_LOW_PASS && !EEPROM_LOW_PASS) {         // якщо дозволена автонастройка
нижнього порога шумів
    autoLowPass();
  }
  if (EEPROM_LOW_PASS) {                // відновити значення шумів з пам'яті
    LOW_PASS = EEPROM.readInt(70);
    SPEKTR_LOW_PASS = EEPROM.readInt(72);
  }
  // в 100 комірці зберігається число 100. Якщо ні - значить це перший запуск
системи
  if (KEEP_SETTINGS) {
    if (EEPROM.read(100) != 100) {
      //Serial.println(F("First start"));
      EEPROM.write(100, 100);
      updateEEPROM();
    } else {
      readEEPROM();
    }}
#if (SETTINGS_LOG == 1)
  Serial.print(F("this_mode = ")); Serial.println(this_mode);
  Serial.print(F("freq_strobe_mode = ")); Serial.println(freq_strobe_mode);
  Serial.print(F("light_mode = ")); Serial.println(light_mode);
  Serial.print(F("RAINBOW_STEP = ")); Serial.println(RAINBOW_STEP);
  Serial.print(F("MAX_COEF_FREQ = ")); Serial.println(MAX_COEF_FREQ);
  Serial.print(F("STROBE_PERIOD = ")); Serial.println(STROBE_PERIOD);
  Serial.print(F("LIGHT_SAT = ")); Serial.println(LIGHT_SAT);
  Serial.print(F("RAINBOW_STEP_2 = ")); Serial.println(RAINBOW_STEP_2);
  Serial.print(F("HUE_START = ")); Serial.println(HUE_START);
  Serial.print(F("SMOOTH = ")); Serial.println(SMOOTH);
  Serial.print(F("SMOOTH_FREQ = ")); Serial.println(SMOOTH_FREQ);
  Serial.print(F("STROBE_SMOOTH = ")); Serial.println(STROBE_SMOOTH);
```

```
  Serial.print(F("LIGHT_COLOR = ")); Serial.println(LIGHT_COLOR);
  Serial.print(F("COLOR_SPEED = ")); Serial.println(COLOR_SPEED);
  Serial.print(F("RAINBOW_PERIOD = ")); Serial.println(RAINBOW_PERIOD);
  Serial.print(F("RUNNING_SPEED = ")); Serial.println(RUNNING_SPEED);
  Serial.print(F("HUE_STEP = ")); Serial.println(HUE_STEP);
  Serial.print(F("EMPTY_BRIGHT = ")); Serial.println(EMPTY_BRIGHT);
  Serial.print(F("ONstate = ")); Serial.println(ONstate);
#endif
}
void loop() {
  buttonTick();     // опитування та обробка кнопки
#if REMOTE_TYPE != 0
  remoteTick();     // опитування ІЧ пульта
#endif
  mainLoop();       // головний цикл обробки і відтворення
  eepromTick();     // перевірка чи не час зберегти настройки
}
void mainLoop() {
  // головний цикл відтворення
  if (ONstate) {
    if (millis() - main_timer > MAIN_LOOP) {
      // скидаємо значення
      RsoundLevel = 0;
      LsoundLevel = 0;
      // перші два режими - гучність (VU meter)
      if (this_mode == 0 || this_mode == 1) {
        for (byte i = 0; i < 100; i ++) {                                  //
робимо 100 вимірювань
          RcurrentLevel = analogRead(SOUND_R);                             // з
правого
          if (!MONO) LcurrentLevel = analogRead(SOUND_L);                  // і
лівого каналів
          if (RsoundLevel < RcurrentLevel) RsoundLevel = RcurrentLevel;    //
шукаємо максимальне
          if (!MONO) if (LsoundLevel < LcurrentLevel) LsoundLevel = LcurrentLevel;
// шукаємо максимальне
        }
        // фільтруємо по нижньому порозі шумів
        RsoundLevel = map(RsoundLevel, LOW_PASS, 1023, 0, 500);
        if (!MONO)LsoundLevel = map(LsoundLevel, LOW_PASS, 1023, 0, 500);
        // обмежуємо діапазон
        RsoundLevel = constrain(RsoundLevel, 0, 500);
        if (!MONO)LsoundLevel = constrain(LsoundLevel, 0, 500);
        // зводимо в ступінь (для більшої чіткості роботи)
        RsoundLevel = pow(RsoundLevel, EXP);
        if (!MONO)LsoundLevel = pow(LsoundLevel, EXP);
        // фільтр
        RsoundLevel_f = RsoundLevel * SMOOTH + RsoundLevel_f * (1 - SMOOTH);
        if (!MONO)LsoundLevel_f = LsoundLevel * SMOOTH + LsoundLevel_f * (1 -
SMOOTH);
        if (MONO) LsoundLevel_f = RsoundLevel_f;  // якщо моно, то лівий = правому
```

```
        // заливаємо "підкладку", якщо яскравість достатня
        if (EMPTY_BRIGHT > 5) {
          for (int i = 0; i < NUM_LEDS; i++)
            leds[i] = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
        }
        // якщо значення вище порогового - починаємо
        if (RsoundLevel_f > 15 && LsoundLevel_f > 15) {
          // розрахунок загальної середньої гучності з обох каналів, фільтрація.
          // Фільтр дуже повільний, зроблено спеціально для автогучності
          averageLevel = (float)(RsoundLevel_f + LsoundLevel_f) / 2 * averK +
averageLevel * (1 - averK);
          // беремо максимальну гучність шкали як середню, помножену на деякий
коефіцієнт MAX_COEF
          maxLevel = (float)averageLevel * MAX_COEF;
          // перетворимо сигнал в довжину стрічки (де MAX_CH це половина кількості
світлодіодів)
          Rlenght = map(RsoundLevel_f, 0, maxLevel, 0, MAX_CH);
          Llenght = map(LsoundLevel_f, 0, maxLevel, 0, MAX_CH);
          // обмежуємо до макс. числа світлодіодів
          Rlenght = constrain(Rlenght, 0, MAX_CH);
          Llenght = constrain(Llenght, 0, MAX_CH);
          animation();       // відтворити
        }}
      // 3-5 режим - світломузика
      if (this_mode == 2 || this_mode == 3 || this_mode == 4 || this_mode == 7 ||
this_mode == 8) {
        analyzeAudio();
        colorMusic[0] = 0;
        colorMusic[1] = 0;
        colorMusic[2] = 0;
        for (int i = 0 ; i < 32 ; i++) {
          if (fht_log_out[i] < SPEKTR_LOW_PASS) fht_log_out[i] = 0;
        }
        // низькі частоти, вибірка з 2 по 5 тон (0 і 1 зашумлені!)
        for (byte i = 2; i < 6; i++) {
          if (fht_log_out[i] > colorMusic[0]) colorMusic[0] = fht_log_out[i];
        }
        // середні частоти, вибірка з 6 по 10 тон
        for (byte i = 6; i < 11; i++) {
          if (fht_log_out[i] > colorMusic[1]) colorMusic[1] = fht_log_out[i];
        }
        // високі частоти, вибірка з 11 по 31 тон
        for (byte i = 11; i < 32; i++) {
          if (fht_log_out[i] > colorMusic[2]) colorMusic[2] = fht_log_out[i];
        }
        freq_max = 0;
        for (byte i = 0; i < 30; i++) {
          if (fht_log_out[i + 2] > freq_max) freq_max = fht_log_out[i + 2];
          if (freq_max < 5) freq_max = 5;
          if (freq_f[i] < fht_log_out[i + 2]) freq_f[i] = fht_log_out[i + 2];
          if (freq_f[i] > 0) freq_f[i] -= LIGHT_SMOOTH;
```

```
          else freq_f[i] = 0;
        }
        freq_max_f = freq_max * averK + freq_max_f * (1 - averK);
        for (byte i = 0; i < 3; i++) {
          colorMusic_aver[i] = colorMusic[i] * averK + colorMusic_aver[i] * (1 -
averK);  // загальна фільтрація
          colorMusic_f[i] = colorMusic[i] * SMOOTH_FREQ + colorMusic_f[i] * (1 -
SMOOTH_FREQ);      // локальна
          if (colorMusic_f[i] > ((float)colorMusic_aver[i] * MAX_COEF_FREQ)) {
            thisBright[i] = 255;
            colorMusicFlash[i] = true;
            running_flag[i] = true;
          } else colorMusicFlash[i] = false;
          if (thisBright[i] >= 0) thisBright[i] -= SMOOTH_STEP;
          if (thisBright[i] < EMPTY_BRIGHT) {
            thisBright[i] = EMPTY_BRIGHT;
            running_flag[i] = false;
          }}
        animation();
      }
      if (this_mode == 5) {
        if ((long)millis() - strobe_timer > STROBE_PERIOD) {
          strobe_timer = millis();
          strobeUp_flag = true;
          strobeDwn_flag = false;
        }
        if ((long)millis() - strobe_timer > light_time) {
          strobeDwn_flag = true;
        }
        if (strobeUp_flag) {
          if (strobe_bright < 255)              // якщо яскравість не максимальна
            strobe_bright += STROBE_SMOOTH;     // збільшити
          if (strobe_bright > 255) {            // якщо пробили макс. яскравість
            strobe_bright = 255;                // залишити максимум
            strobeUp_flag = false;              // прапорець відпустити
          }}
        if (strobeDwn_flag) {                   // гаснемо
          if (strobe_bright > 0)                // якщо яскравість не мінімальна
            strobe_bright -= STROBE_SMOOTH;     // зменшити
          if (strobe_bright < 0) {              // якщо пробили мін. яскравість
            strobeDwn_flag = false;
            strobe_bright = 0;                  // залишити 0
          }}
        animation();
      }
      if (this_mode == 6) animation();
      if (!IRLremote.receiving())    // якщо на ІЧ приймач не потрапляє сигнал
(без цього НЕ ПРАЦЮЄ!)
        FastLED.show();         // відправити значення на стрічку
      if (this_mode != 7)       // 7 режиму не потрібне очищення !!!
        FastLED.clear();          // очистити масив пікселів
```

```
      main_timer = millis();     // скинути таймер
    }}}
void animation() {
  // згідно з режимом
  switch (this_mode) {
    case 0:
      count = 0;
      for (int i = (MAX_CH - 1); i > ((MAX_CH - 1) - Rlenght); i--) {
        leds[i] = ColorFromPalette(myPal, (count * index));    // заливка по
палітрі "від зеленого до червоного"
        count++;
      }
      count = 0;
      for (int i = (MAX_CH); i < (MAX_CH + Llenght); i++ ) {
        leds[i] = ColorFromPalette(myPal, (count * index));    // заливка по
палітрі "від зеленого до червоного"
        count++;
      }
      if (EMPTY_BRIGHT > 0) {
        CHSV this_dark = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
        for (int i = ((MAX_CH - 1) - Rlenght); i > 0; i--)
          leds[i] = this_dark;
        for (int i = MAX_CH + Llenght; i < NUM_LEDS; i++)
          leds[i] = this_dark;
      }
      break;
    case 1:
      if (millis() - rainbow_timer > 30) {
        rainbow_timer = millis();
        hue = floor((float)hue + RAINBOW_STEP);
      }
      count = 0;
      for (int i = (MAX_CH - 1); i > ((MAX_CH - 1) - Rlenght); i--) {
        leds[i] = ColorFromPalette(RainbowColors_p, (count * index) / 2 - hue);
// заливка по палітрі веселка
        count++;
      }
      count = 0;
      for (int i = (MAX_CH); i < (MAX_CH + Llenght); i++ ) {
        leds[i] = ColorFromPalette(RainbowColors_p, (count * index) / 2 - hue); //
заливка по палітрі веселка
        count++;
      }
      if (EMPTY_BRIGHT > 0) {
        CHSV this_dark = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
        for (int i = ((MAX_CH - 1) - Rlenght); i > 0; i--)
          leds[i] = this_dark;
        for (int i = MAX_CH + Llenght; i < NUM_LEDS; i++)
          leds[i] = this_dark;
      }
      break;
```

```
    case 2:
      for (int i = 0; i < NUM_LEDS; i++) {
        if (i < STRIPE)          leds[i] = CHSV(HIGH_COLOR, 255, thisBright[2]);
        else if (i < STRIPE * 2) leds[i] = CHSV(MID_COLOR, 255, thisBright[1]);
        else if (i < STRIPE * 3) leds[i] = CHSV(LOW_COLOR, 255, thisBright[0]);
        else if (i < STRIPE * 4) leds[i] = CHSV(MID_COLOR, 255, thisBright[1]);
        else if (i < STRIPE * 5) leds[i] = CHSV(HIGH_COLOR, 255, thisBright[2]);
      }
      break;
    case 3:
      for (int i = 0; i < NUM_LEDS; i++) {
        if (i < NUM_LEDS / 3)          leds[i] = CHSV(HIGH_COLOR, 255,
thisBright[2]);
        else if (i < NUM_LEDS * 2 / 3) leds[i] = CHSV(MID_COLOR, 255,
thisBright[1]);
        else if (i < NUM_LEDS)         leds[i] = CHSV(LOW_COLOR, 255,
thisBright[0]);
      }
      break;
    case 4:
      switch (freq_strobe_mode) {
        case 0:
          if (colorMusicFlash[2]) HIGHS();
          else if (colorMusicFlash[1]) MIDS();
          else if (colorMusicFlash[0]) LOWS();
          else SILENCE();
          break;
        case 1:
          if (colorMusicFlash[2]) HIGHS();
          else SILENCE();
          break;
        case 2:
          if (colorMusicFlash[1]) MIDS();
          else SILENCE();
          break;
        case 3:
          if (colorMusicFlash[0]) LOWS();
          else SILENCE();
          break;
      }
      break;
    case 5:
      if (strobe_bright > 0)
        for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(STROBE_COLOR,
STROBE_SAT, strobe_bright);
      else
        for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(EMPTY_COLOR, 255,
EMPTY_BRIGHT);
      break;
    case 6:
      switch (light_mode) {
```

```
        case 0: for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(LIGHT_COLOR,
LIGHT_SAT, 255);
          break;
        case 1:
          if (millis() - color_timer > COLOR_SPEED) {
            color_timer = millis();
            if (++this_color > 255) this_color = 0;
          }
          for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(this_color, LIGHT_SAT,
255);
          break;
        case 2:
          if (millis() - rainbow_timer > 30) {
            rainbow_timer = millis();
            this_color += RAINBOW_PERIOD;
            if (this_color > 255) this_color = 0;
            if (this_color < 0) this_color = 255;
          }
          rainbow_steps = this_color;
          for (int i = 0; i < NUM_LEDS; i++) {
            leds[i] = CHSV((int)floor(rainbow_steps), 255, 255);
            rainbow_steps += RAINBOW_STEP_2;
            if (rainbow_steps > 255) rainbow_steps = 0;
            if (rainbow_steps < 0) rainbow_steps = 255;
          }
          break;
      }
      break;
    case 7:
      switch (freq_strobe_mode) {
        case 0:
          if (running_flag[2]) leds[NUM_LEDS / 2] = CHSV(HIGH_COLOR, 255,
thisBright[2]);
          else if (running_flag[1]) leds[NUM_LEDS / 2] = CHSV(MID_COLOR, 255,
thisBright[1]);
          else if (running_flag[0]) leds[NUM_LEDS / 2] = CHSV(LOW_COLOR, 255,
thisBright[0]);
          else leds[NUM_LEDS / 2] = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
          break;
        case 1:
          if (running_flag[2]) leds[NUM_LEDS / 2] = CHSV(HIGH_COLOR, 255,
thisBright[2]);
          else leds[NUM_LEDS / 2] = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
          break;
        case 2:
          if (running_flag[1]) leds[NUM_LEDS / 2] = CHSV(MID_COLOR, 255,
thisBright[1]);
          else leds[NUM_LEDS / 2] = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
          break;
        case 3:
```

```
          if (running_flag[0]) leds[NUM_LEDS / 2] = CHSV(LOW_COLOR, 255,
thisBright[0]);
          else leds[NUM_LEDS / 2] = CHSV(EMPTY_COLOR, 255, EMPTY_BRIGHT);
          break;
      }
      leds[(NUM_LEDS / 2) - 1] = leds[NUM_LEDS / 2];
      if (millis() - running_timer > RUNNING_SPEED) {
        running_timer = millis();
        for (int i = 0; i < NUM_LEDS / 2 - 1; i++) {
          leds[i] = leds[i + 1];
          leds[NUM_LEDS - i - 1] = leds[i];
        }}
      break;
    case 8:
      byte HUEindex = HUE_START;
      for (int i = 0; i < NUM_LEDS / 2; i++) {
        byte this_bright = map(freq_f[(int)floor((NUM_LEDS / 2 - i) /
freq_to_stripe)], 0, freq_max_f, 0, 255);
        this_bright = constrain(this_bright, 0, 255);
        leds[i] = CHSV(HUEindex, 255, this_bright);
        leds[NUM_LEDS - i - 1] = leds[i];
        HUEindex += HUE_STEP;
        if (HUEindex > 255) HUEindex = 0;
      }
      break;
  }}
void HIGHS() {
  for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(HIGH_COLOR, 255,
thisBright[2]);
}
void MIDS() {
  for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(MID_COLOR, 255,
thisBright[1]);
}
void LOWS() {
  for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(LOW_COLOR, 255,
thisBright[0]);
}
void SILENCE() {
  for (int i = 0; i < NUM_LEDS; i++) leds[i] = CHSV(EMPTY_COLOR, 255,
EMPTY_BRIGHT);
}
// допоміжна функція, змінює величину value на крок incr в межах minimum ..
maximum
int smartIncr(int value, int incr_step, int mininmum, int maximum) {
  int val_buf = value + incr_step;
  val_buf = constrain(val_buf, mininmum, maximum);
  return val_buf;
}
float smartIncrFloat(float value, float incr_step, float mininmum, float maximum)
{
```

```
  float val_buf = value + incr_step;
  val_buf = constrain(val_buf, mininmum, maximum);
  return val_buf;
}
#if REMOTE_TYPE != 0
void remoteTick() {
  if (Serial.available())  {
    IRdata = Serial.readString();
    ir_flag = true;
  }
  if (ir_flag) { // якщо дані прийшли
    eeprom_timer = millis();
    eeprom_flag = true;
    // режимы
    if (IRdata == BUTT_1) {
      this_mode = 0;}
    else if ((IRdata == BUTT_2)){
    this_mode = 1;}
    else if ((IRdata == BUTT_3)){
    this_mode = 2;}
    else if ((IRdata == BUTT_4)){
    this_mode = 3;}
    else if ((IRdata == BUTT_5)){
    this_mode = 4;}
    else if ((IRdata == BUTT_6)){
    this_mode = 5;}
    else if ((IRdata == BUTT_7)){
    this_mode = 6;}
    else if ((IRdata == BUTT_8)){
    this_mode = 7;}
    else if ((IRdata == BUTT_9)){
    this_mode = 8;}
    else if ((IRdata == BUTT_0)){
    fullLowPass();}
    else if ((IRdata == BUTT_STAR)){
    ONstate = !ONstate; FastLED.clear(); FastLED.show(); updateEEPROM();}
    else if ((IRdata == BUTT_HASH)){
    switch (this_mode) {
          case 4:
          case 7: if (++freq_strobe_mode > 3) freq_strobe_mode = 0;
            break;
          case 6: if (++light_mode > 2) light_mode = 0;
            break;
        }}
     else if ((IRdata == BUTT_OK)){
     digitalWrite(MLED_PIN, settings_mode ^ MLED_ON); settings_mode =
!settings_mode;}
     else if ((IRdata == BUTT_UP)){
     if (settings_mode) {
          // ВВЕРХ загальні налаштування
          EMPTY_BRIGHT = smartIncr(EMPTY_BRIGHT, 5, 0, 255);
```

```
        } else {
          switch (this_mode) {
            case 0:
              break;
            case 1: RAINBOW_STEP = smartIncrFloat(RAINBOW_STEP, 0.5, 0.5, 20);
              break;
            case 2:
            case 3:
            case 4: MAX_COEF_FREQ = smartIncrFloat(MAX_COEF_FREQ, 0.1, 0, 5);
              break;
            case 5: STROBE_PERIOD = smartIncr(STROBE_PERIOD, 20, 1, 1000);
              break;
            case 6:
              switch (light_mode) {
                case 0: LIGHT_SAT = smartIncr(LIGHT_SAT, 20, 0, 255);
                  break;
                case 1: LIGHT_SAT = smartIncr(LIGHT_SAT, 20, 0, 255);
                  break;
                case 2: RAINBOW_STEP_2 = smartIncrFloat(RAINBOW_STEP_2, 0.5, 0.5,
10);
                  break;
              }
              break;
            case 7: MAX_COEF_FREQ = smartIncrFloat(MAX_COEF_FREQ, 0.1, 0.0, 10);
              break;
            case 8: HUE_START = smartIncr(HUE_START, 10, 0, 255);
              break;
          }}}
      else if ((IRdata == BUTT_DOWN)){
      if (settings_mode) {
          // ВНИЗ загальні налаштування
          EMPTY_BRIGHT = smartIncr(EMPTY_BRIGHT, -5, 0, 255);
        } else {
          switch (this_mode) {
            case 0:
              break;
            case 1: RAINBOW_STEP = smartIncrFloat(RAINBOW_STEP, -0.5, 0.5, 20);
              break;
            case 2:
            case 3:
            case 4: MAX_COEF_FREQ = smartIncrFloat(MAX_COEF_FREQ, -0.1, 0, 5);
              break;
            case 5: STROBE_PERIOD = smartIncr(STROBE_PERIOD, -20, 1, 1000);
              break;
            case 6:
              switch (light_mode) {
                case 0: LIGHT_SAT = smartIncr(LIGHT_SAT, -20, 0, 255);
                  break;
                case 1: LIGHT_SAT = smartIncr(LIGHT_SAT, -20, 0, 255);
                  break;
```

```
                case 2: RAINBOW_STEP_2 = smartIncrFloat(RAINBOW_STEP_2, -0.5, 0.5,
10);
                  break;
                }
                break;
              case 7: MAX_COEF_FREQ = smartIncrFloat(MAX_COEF_FREQ, -0.1, 0.0, 10);
                break;
              case 8: HUE_START = smartIncr(HUE_START, -10, 0, 255);
                break;
            }}}
      else if ((IRdata == BUTT_LEFT)){
       if (settings_mode) {
          // ВЛІВО загальні налаштування
          BRIGHTNESS = smartIncr(BRIGHTNESS, -20, 0, 255);
          FastLED.setBrightness(BRIGHTNESS);
        } else {
          switch (this_mode) {
            case 0:
            case 1: SMOOTH = smartIncrFloat(SMOOTH, -0.05, 0.05, 1);
              break;
            case 2:
            case 3:
            case 4: SMOOTH_FREQ = smartIncrFloat(SMOOTH_FREQ, -0.05, 0.05, 1);
              break;
            case 5: STROBE_SMOOTH = smartIncr(STROBE_SMOOTH, -20, 0, 255);
              break;
            case 6:
              switch (light_mode) {
                case 0: LIGHT_COLOR = smartIncr(LIGHT_COLOR, -10, 0, 255);
                  break;
                case 1: COLOR_SPEED = smartIncr(COLOR_SPEED, -10, 0, 255);
                  break;
                case 2: RAINBOW_PERIOD = smartIncr(RAINBOW_PERIOD, -1, -20, 20);
                  break;
              }
              break;
            case 7: RUNNING_SPEED = smartIncr(RUNNING_SPEED, -10, 1, 255);
              break;
            case 8: HUE_STEP = smartIncr(HUE_STEP, -1, 1, 255);
              break;
          }}}
      else if ((IRdata == BUTT_RIGHT)){
      if (settings_mode) {
          // ВПРАВО загальні налаштування
          BRIGHTNESS = smartIncr(BRIGHTNESS, 20, 0, 255);
          FastLED.setBrightness(BRIGHTNESS);
        } else {
          switch (this_mode) {
            case 0:
            case 1: SMOOTH = smartIncrFloat(SMOOTH, 0.05, 0.05, 1);
              break;
```

```
            case 2:
            case 3:
            case 4: SMOOTH_FREQ = smartIncrFloat(SMOOTH_FREQ, 0.05, 0.05, 1);
              break;
            case 5: STROBE_SMOOTH = smartIncr(STROBE_SMOOTH, 20, 0, 255);
              break;
            case 6:
              switch (light_mode) {
                case 0: LIGHT_COLOR = smartIncr(LIGHT_COLOR, 10, 0, 255);
                  break;
                case 1: COLOR_SPEED = smartIncr(COLOR_SPEED, 10, 0, 255);
                  break;
                case 2: RAINBOW_PERIOD = smartIncr(RAINBOW_PERIOD, 1, -20, 20);
                  break;
              }
              break;
            case 7: RUNNING_SPEED = smartIncr(RUNNING_SPEED, 10, 1, 255);
              break;
            case 8: HUE_STEP = smartIncr(HUE_STEP, 1, 1, 255);
              break;
          }}}
      else{
      eeprom_flag = false;}// якщо не розпізнали кнопку, що не оновлюємо
настройки!
    }
    ir_flag = false;
  }
#endif
void autoLowPass() {
  // для режиму VU
  delay(10);                                // чекаємо ініціалізації АЦП
  int thisMax = 0;                          // максимум
  int thisLevel;
  for (byte i = 0; i < 200; i++) {
    thisLevel = analogRead(SOUND_R);        // робимо 200 вимірювань
    if (thisLevel > thisMax)                // шукаємо максимуми
      thisMax = thisLevel;                  // запам'ятовуємо
    delay(4);                               // чекаємо 4мс
  }
  LOW_PASS = thisMax + LOW_PASS_ADD;        // нижній поріг як максимум тиші +
певна величина
  // для режиму спектра
  thisMax = 0;
  for (byte i = 0; i < 100; i++) {          // робимо 100 вимірювань
    analyzeAudio();                         // розбити в спектр
    for (byte j = 2; j < 32; j++) {         // перші 2 канали - мотлох
      thisLevel = fht_log_out[j];
      if (thisLevel > thisMax)              // шукаємо максимуми
        thisMax = thisLevel;                // запам'ятовуємо
    }
    delay(4);                               // чекаємо 4мс
```

```
  }
  SPEKTR_LOW_PASS = thisMax + LOW_PASS_FREQ_ADD;  // нижній поріг як максимум тиші
  if (EEPROM_LOW_PASS && !AUTO_LOW_PASS) {
    EEPROM.updateInt(70, LOW_PASS);
    EEPROM.updateInt(72, SPEKTR_LOW_PASS);
  }}
void analyzeAudio() {
  for (int i = 0 ; i < FHT_N ; i++) {
    int sample = analogRead(SOUND_R_FREQ);
    fht_input[i] = sample; // поставити реальні дані
  }
  fht_window();  // вікно даних для кращої частотної характеристики
  fht_reorder(); // впорядкувати дані перед тим, як зробити FHT
  fht_run();     // обробляти дані у FHT
  fht_mag_log(); // взяти вихід FHT
}
void buttonTick() {
  butt1.tick();  // обов'язкова функція відпрацювання. Повинна постійно
опитуватись
  if (butt1.isSingle())                              // якщо одиничне натискання
    if (++this_mode >= MODE_AMOUNT) this_mode = 0;    // змінити режим
  if (butt1.isHolded()) {     // кнопка утримана
    fullLowPass();
  }}
void fullLowPass() {
  digitalWrite(MLED_PIN, MLED_ON);   // включити світлодіод
  FastLED.setBrightness(0); // погасити стрічку
  FastLED.clear();          // очистити масив пікселів
  FastLED.show();           // відправити значення на стрічку
  delay(500);               // зачекати трохи
  autoLowPass();            // виміряти шуми
  delay(500);               // зачекати трохи
  FastLED.setBrightness(BRIGHTNESS);  // повернути яскравість
  digitalWrite(MLED_PIN, !MLED_ON);    // вимкнути світлодіод
}
void updateEEPROM() {
  EEPROM.updateByte(1, this_mode);
  EEPROM.updateByte(2, freq_strobe_mode);
  EEPROM.updateByte(3, light_mode);
  EEPROM.updateInt(4, RAINBOW_STEP);
  EEPROM.updateFloat(8, MAX_COEF_FREQ);
  EEPROM.updateInt(12, STROBE_PERIOD);
  EEPROM.updateInt(16, LIGHT_SAT);
  EEPROM.updateFloat(20, RAINBOW_STEP_2);
  EEPROM.updateInt(24, HUE_START);
  EEPROM.updateFloat(28, SMOOTH);
  EEPROM.updateFloat(32, SMOOTH_FREQ);
  EEPROM.updateInt(36, STROBE_SMOOTH);
  EEPROM.updateInt(40, LIGHT_COLOR);
  EEPROM.updateInt(44, COLOR_SPEED);
  EEPROM.updateInt(48, RAINBOW_PERIOD);
```

```
  EEPROM.updateInt(52, RUNNING_SPEED);
  EEPROM.updateInt(56, HUE_STEP);
  EEPROM.updateInt(60, EMPTY_BRIGHT);
  if (KEEP_STATE) EEPROM.updateByte(64, ONstate);
}
void readEEPROM() {
  this_mode = EEPROM.readByte(1);
  freq_strobe_mode = EEPROM.readByte(2);
  light_mode = EEPROM.readByte(3);
  RAINBOW_STEP = EEPROM.readInt(4);
  MAX_COEF_FREQ = EEPROM.readFloat(8);
  STROBE_PERIOD = EEPROM.readInt(12);
  LIGHT_SAT = EEPROM.readInt(16);
  RAINBOW_STEP_2 = EEPROM.readFloat(20);
  HUE_START = EEPROM.readInt(24);
  SMOOTH = EEPROM.readFloat(28);
  SMOOTH_FREQ = EEPROM.readFloat(32);
  STROBE_SMOOTH = EEPROM.readInt(36);
  LIGHT_COLOR = EEPROM.readInt(40);
  COLOR_SPEED = EEPROM.readInt(44);
  RAINBOW_PERIOD = EEPROM.readInt(48);
  RUNNING_SPEED = EEPROM.readInt(52);
  HUE_STEP = EEPROM.readInt(56);
  EMPTY_BRIGHT = EEPROM.readInt(60);
  if (KEEP_STATE) ONstate = EEPROM.readByte(64);
}
void eepromTick() {
  if (eeprom_flag)
    if (millis() - eeprom_timer > 30000) {   // 30 секунд після останнього
натискання з пульта
      eeprom_flag = false;
      eeprom_timer = millis();
      updateEEPROM();
    }}
```