НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні мережі та системи”

спеціальності 123 “Комп’ютерна інженерія”

на тему: *Засоби віддаленої розробки та налагодження вбудованих систем*

Виконав : студентКА __4__ курсу, групи _ІВ-93
(шифр групи)

Слюсарь Регіна Олександрівна
(прізвище, ім’я, по батькові) (підпис)

Керівник _доц. каф. ОТ, к.т.н., доц.Ткаченко В.В
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) старший викладач Виноградов Ю.М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп’ютерні мережі та системи ”

спеціальності 123 “Комп’ютерна інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Слюсарь Регіни Олександрівни

1. Тема проєкту *Засоби віддаленої розробки та налагодження вбудованих систем*
керівник проєкту Ткаченко Валентина Василівна, доц. каф. ОТ, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №1139-с

2. Термін здачі студентом закінченого проєкту *8 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Теоритичні аспекти виддаленої розробки та наголодження вбудованих систем.

Розділ 2. Огляд існуючих рішень віддаленої розробки та наголодження вбудованих систем.

Розділ 3. Вибір апаратних та програмних ресурсів:обґрунтування та аналіз.

Розділ 4. Налаштування і використання віддаленого доступу до вбудованих систем.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) алгоритм отримання віддаленого доступу до вбудовоної системи, процес взаємодії між компонентами системи, .

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «31» серпня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*13.11.2022-26.02.2023*	
2.	*Вивчення та аналіз завдання*	*26.02.2023-30.04.2023*	
3.	*Розробка архітектури та загальної структури системи*	*30.04.2023-07.05.2023*	
4.	*Розробка структур окремих підсистем*	*07.05.2023-21.05.2023*	
5.	*Програмна реалізація системи*	*14.05.2023-21.05.2023*	
6.	*Оформлення пояснювальної записки*	*27.05.2023*	
7.	*Захист програмного продукту*	*05.06.2023*	
8.	*Передзахист*	*08.06.2023*	
9.	*Захист*	*19.06.2023*	

Студент-дипломник _____ Регіна СЛЮСАРЬ
(підпис)

Керівник проєкту _____ Валентина ТКАЧЕНКО
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто принципи та методи віддаленої розробки та налагодження вбудованих систем. Були проаналізовані їхні можливості, обмеження та практичні застосування. Було вибрано та обґрунтовано використання специфічних засобів для віддаленої розробки та налагодження вбудованих систем. В ході роботи були налаштовані та продемонстровані такі технології, як SSH, VNC, HTTP-сервер, хмарні сервіси AWS. Розроблені методики та процедури можуть бути використані для віддаленої розробки, налагодження та моніторингу різних вбудованих систем, що дозволяє збільшити гнучкість та ефективність процесу розробки. Вся розробка та демонстрації були проведені за допомогою BeagleBone Black в якості вбудованої системи.

Ключові слова: вбудовані системи, віддалена розробка, налагодження, SSH, VNC, HTTP-сервер, AWS, BeagleBone Black.

ANNOTATION

In this thesis for a Bachelor's Degree, the principles and methods of remote development and debugging of embedded systems were considered in detail. Their capabilities, limitations, and practical applications were analyzed. Specific tools for remote development and debugging of embedded systems were selected and justified. During the work, technologies such as SSH, VNC, HTTP server, AWS cloud services were configured and demonstrated. The developed methodologies and procedures can be used for remote development, debugging, and monitoring of various embedded systems, increasing the flexibility and efficiency of the development process. All development and demonstrations were performed using BeagleBone Black as the embedded system.

Key words: embedded systems, remote development, debugging, SSH, VNC, HTTP server, AWS, BeagleBone Black.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Засоби віддаленої розробки	4		
			та налагодження			
			вбудованих систем			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Засоби віддаленої розробки	103		
			та налагодження вс			
			вбудованих систем			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Засоби віддаленої розробки	1		
			та налагодження вс			
			Алгоритм отримання			
			віддаленого доступу до вс			
			Структурна схема			
	А4	***ІАЛЦ.4672008.005 Д2***	Засоби віддаленої	1		
			розробки та налагодження			
			вбудованих систем			
			Процес взаємодії між			
			компонентами системи			
			Функціональна схема			
	А4	***ІАЛЦ.467200.006 Д3***	Засоби віддаленої	1		
			розробки та налагодження			
			вбудованих систем			
			Схема підключення			
			Вбудованої системи			
	А4	***ІАЛЦ.467200.007 Д4***	Засоби віддаленої	9		
			розробки та налагодження			
			вбудованих систем			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	***ІАЛЦ.467200.001 ОА***			
Розроб		Слюсарь Р.О.			*Засоби віддаленої розробки та налагодження вбудованих систем*	Літ.	Аркуш	Аркушів
Перев		Ткаченко В.В.					1	1
					Опис альбому	***НТУУ "КПІ" ФІОТ ІВ-93***		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Засоби віддаленої розробки та налагодження вбудованих систем*»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання включає розробку засобів для віддаленої розробки та налагодження вбудованих систем, які працюють на платформі BeagleBone Black.

Область застосування включає в себе вбудовані системи для промислових, телекомунікаційних, автомобільних, аерокосмічних та медичних застосувань.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп' ютерної інженерії», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка засобів для віддаленої розробки та налагодження вбудованих систем на платформі BeagleBone Black, що дозволить інженерам програмного забезпечення віддалено розробляти, відлагоджувати та виконувати тестування вбудованих систем, забезпечуючи ефективність та продуктивність процесу розробки.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, технологій, які використовувались для розробки, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість інженерам програмного забезпечення віддалено розробляти, відлагоджувати та виконувати тестування вбудованих систем.
- Обов'язковий доступ до Інтернету для виконання віддалених операцій.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.
- Надати засоби для моніторингу та управління віддаленими пристроями.

5.2. Вимоги до програмного забезпечення

- ОС Mac, Linux чи Windows.
- Веб-браузер останньої версії.

5.3. Вимоги до апаратної частини

- BeagleBone Black плата для вбудованих систем.
- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.
- TTL кабель для з'єднання та комунікації з BeagleBone Black.
- Ethernet кабель для підключення до мережі Інтернет.
- MiniUSB для живлення BeagleBone Black.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	12.12.2022-15.12.2022
Вивчення та аналіз завдання	15.03.2023-25.03.2023
Розробка архітектури та загальної структури системи	26.04.2023-30.04.2023
Розробка структур окремих частин системи	01.04.2023-30.04.2023
Програмна реалізація системи	7.05.2023-14.05.2023
Виправлення помилок	14.05.2023
Оформлення пояснювальної записки	15.05.2023-27.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Засоби віддаленої розробки та налагодження вбудованих систем*»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SSH	(Secure Shell) Безпечна оболонка
VNC	(Virtual Network Computering) Віртуальна мережа обчислень
AWS	(Amazon Web Services) Веб-сервіси Амазон
VPN	(Virtual Private Network) Віртуальна приватна мережа
HTTP	(Hepertext Transfer Protocol) Протокол передачі гіпертексту
ПК	Персональний Комп’ютер
ОС	Оперативна Система
IDE	(Integrated Development enviroment) Інтегроване середовище розробки
LAN	(Invasive Weed Optimization) Локальна область мережі
JTA	(Joint Technical Architecture) Спільна технічна архітектура
USB	(Universal Serial Bus) Універсальна послідовна шина)
TTL	(Transistor-Transistor Logic) Транзистор-транзисторна логіка
TCP	(Transmission Control Protocol) Протокол контролю передачі
IP	(Internet Protocol) Інтернет-протокол
JTAG	(Joint Test Action Group) Спільна група тестування
HDMI	(High-Definition Multimedia Inrerface) Інтерфейс мультимедіа високої чіткості
RFB	(Remote Framebuffer) Віддалений кадр буфера

ВСТУП

З розширенням технологічних можливостей і складності програмного забезпечення, набуває важливості проблема віддаленої розробки та налагодження таких систем. Враховуючи особливості вбудованих систем, а саме обмеженість ресурсів, високі вимоги до надійності та часто необхідність постійної роботи, виникає потреба в пошуку нових підходів та інструментів для їх розробки та підтримки. Тож усе частіше, інженери та розробники по всьому світу працюють над створенням та налагодженням вбудованих систем віддалено, використовуючи різноманітні інструменти та сервіси, що дозволяють ефективно розв'язувати поставлені завдання.

Вбудовані системи є невід'ємною частиною багатьох сфер нашого життя - від промислових установок та медичних приладів до домашньої електроніки та автомобільних систем. Всі ці системи вимагають точної розробки, налагодження та підтримки, що вимагає від розробників не тільки глибоких знань у своїй області, але і володіння сучасними засобами та технологіями, що дозволяють ефективно виконувати ці завдання.

Віддалена розробка та налагодження вбудованих систем є важливою складовою їх підтримки, адже вбудовані системи часто знаходяться в умовах, коли прямий доступ до них обмежений або неможливий. Тож віддалене налагодження дозволяє збільшити продуктивність розробників, що знаходяться не на місці розміщення пристрою.

В даній роботі будуть розглянуті сучасні методи та інструменти для віддаленої розробки та налагодження вбудованих систем, а також їх переваги та недоліки. Також будуть висвітлені актуальні технології, включаючи SSH, VNC, HTTP-сервери, хмарні сервіси, такі як AWS. Особливу увагу буде приділено практичним аспектам використання цих технологій в реальних умовах розробки та налагодження вбудованих систем.

РОЗДІЛ І. ТЕОРЕТИЧНІ АСПЕКТИ ВІДДАЛЕНОЇ РОЗРОБКИ ТА НАЛАГОДЖЕННЯ ВБУДОВАНИХ СИСТЕМ

1.1 Основні поняття вбудованих систем

1.1.1 Розгляд визначень вбудованої системи

Визначення «вбудованої системи» є змінним, воно змінюється разом із прогресом технологій, а також зі зміною вартості впровадження апаратних та програмних компонентів.

Розглянемо кілька найпоширеніших описів вбудованої системи:

- Вбудовані системи - це розумний пристрій із процесором, який має більш обмежені функції апаратного та/або програмного забезпечення, ніж персональний комп'ютер (ПК). Це справедливо для значної частини комп'ютерних вбудованих систем. Апаратні обмеження можуть стосуватися : обмеження продуктивності обробки, енергоспоживання, пам'яті, функціональності апаратного забезпечення тощо. У програмному забезпеченні це зазвичай означає обмеження щодо ПК: менше додатків, відсутність операційної системи (ОС) або обмежена ОС або код з меншим рівнем абстракції.

Типи програмних функцій, які процесор може виконувати у вбудованій системі:

1. Математичні операції та/або операції обробки даних. Він може аналізувати дані та приймати рішення на основі даних.
2. Обробка та керування часом: як вхід (наприклад, період вимірювання), вихід (наприклад, вихідні сигнали) і засіб для синхронізації завдань (наприклад, виконання 1000 разів на секунду).

3. Введення/виведення в реальному часі з метою вимірювання або контролю.
4. Цифрова обробка сигналів (DSP), яка є математичними обчисленнями над потоками даних. Приклади включають аудіо, відео, радар і гідролокатор.
5. Зв'язок і мережа.

Однак, вже на сьогоднішній час, це визначення лише частково вірне, оскільки плати та програмне забезпечення, які зазвичай використовувалися в комп’ютерах минулого, були перекомпоновані у складніші конструкції вбудованих систем.

- Вбудована система — це розумний пристрій із процесором, який має спеціальне і особливе призначення. Більшість вбудованих пристроїв в основному розроблено для виконання однієї конкретної функції. Однак зараз ми бачимо пристрої, які призначені для виконання різноманітних основних функцій. На приклад, новітні цифрові телевізори містять інтерактивні додатки, які виконують широкий спектр загальних функцій, не пов'язаних з функцією "TV(телебачення)", таких як електронна пошта, веб-браузинг, ігри та інші.
- Вбудована система — це комп’ютерна система з вищими вимогами до якості та надійності, ніж до інших типів комп’ютерних систем. Деякі сімейства вбудованих пристроїв мають дуже високий поріг вимог до якості, стійкості та надійності. Наприклад, якщо під час під час операції виходить з ладу важливий медичний пристрій, або під час керування літака вийде з ладу система попередження зіткнень, виникають дуже серйозні проблеми, які можуть бути загрозою життя. Однак існують також вбудовані пристрої, такі як телевізори, духовки, посудомийні машини та мобільні телефони, у яких несправність це незручність, але зазвичай не є загрозою для життя.

Існує деяка дискусія щодо того, чи належать комп'ютерні системи, які відповідають деяким, але не всім традиційним визначенням вбудованих систем, саме вбудованим системи чи чомусь іншому. Насправді інженери вбудованих систем розділилися щодо того, чи повинні традиційні визначення вбудованих систем продовжувати розвиватися, чи буде нова область комп’ютерних систем, щоб включати ці більш складні системи.

Тож вбудовані системи не мають конкретної характеристики для єдиного визначення, яке б відображало їх усі. (див. таблицю 1.1).

Табл. 1.1 - Приклади вбудованих систем та їх сегменту

Сігмент	Вбудований девайс
Автомобільний	Система запалювання
	Контроль двигуна
	Гальмівна система
Побутова електроніка	Цифрові та аналогові телевізори
	Приставки
	Асистенти персональних даних
	Кухонна техніка (холодильники, тостери,)
	Автомобілі
	Фотоапарати
	Глобальні системи позиціонування

Продовження Табл. - 1.1.

Автоматизована система управління	Робототехніка та системи управління
Медичний	Інфузійні насоси
	Апарати для діалізу
	Протези
	Кардіомонітори
Нетворкінг	Маршрутизатори
	Шлюзи(Gateways)
Автоматизація офісу	Принтери
	Монітори
	Сканери

1.1.2 Обмеження, які є типовими для вбудованої системи

1. Невеликі розміри вбудованих систем. Багато систем повинні бути ручними.
2. Мала вага. Якщо пристрій розгорнуто в системі, яка рухається, наприклад, прикріплена до людини, літака чи транспортного засобу, то вага спричиняє витрати енергії.
3. Малопотужність вбудованих систем. Вбудованій системі може знадобитися тривалий час працювати від акумулятора. Низька

потужність також впливає на кількість тепла, яке вони можуть генерувати.

4. Стійкість до суворих умов, таких як спека, вода, тиск, вібрація та удари. Вони можуть бути піддані впливу шумів, радіочастотних перешкод, води та хімічних речовин.
5. Надійність при використовуються в критичних для безпеки системах. Важлива поведінка в режимі реального часу. Для цих систем вони повинні належним чином функціонувати з надзвичайно високим рівнем захищеності.
6. Вартость. Більшість програм спрямовані на отримання прибутку. І для систем великого обсягу маленька різниця вартості,може істотно вплинути на прибуток. [1,2,3,5,14]

1.2 Віддалена розробка та налагодження вбудованих систем

Віддалена розробка та налагодження є важливою у сучасному світі вбудованих систем. Ці методи дозволяють розробникам створювати та тестувати програмне забезпечення для пристроїв, які можуть бути недоступні безпосередньо. Є багато варіантів, якими ми можемо віддалено підключатися до вбудованої системи.

1.2.1 Важливість віддаленої розробки

Існує багато причин для застосування методів віддаленої розробки та налагодження. Вбудовані системи часто розміщують у важкодоступних місцях або небезпечних середовищах. Або вбудовані системи виробляються масово, і прямий доступ до кожної одиниці неможливий. Віддалена розробка також може значно пришвидшити цикл розробки, дозволяючи розробляти, обновляти та тестувати в реальному часі без необхідності фізичної взаємодії з пристроєм.

1.2.2 Компоненти віддаленої розробки та налагодження

Віддалена розробка та налагодження вбудованих систем зазвичай вимагають компонентів, таких як аппаратний налагоджувач, віддалений налагоджувач та налагоджувач цілі.

- Аппаратний налагоджувач(Hardware Debugger) - це фізичний пристрій, який підключається до вбудованого пристрою для доступу до його процесора та інших компонентів.
- Віддалений налагоджувач(Remote Debugger) - це програмне забезпечення, яке виконується на розробницькому комп'ютері та комунікує з аппаратним налагоджувачем для надання інтерфейсу для віддаленого налагодження.
- Налагоджувач цілі(Target Debugger) - це програмне забезпечення, яке виконується на вбудованому пристрої та комунікує з віддаленим налагоджувачем, надаючи йому доступ до внутрішніх структур та стану системи.

1.2.3 Процес віддаленої розробки та налагодження

Процес віддаленої розробки та налагодження включає в себе кілька кроків, зазвичай здійснюваних через спеціалізоване середовище розробки (IDE). Після компіляції та збирання коду, він завантажується на вбудований пристрій через аппаратний налагоджувач. За допомогою віддаленого налагоджувача розробник може запускати, зупиняти, крокувати через код, встановлювати точки переривання та відстежувати змінні.

1.2.4 Перешкоди та їх вирішення при віддаленій розроботці та налагодження

Перешкоди, пов'язані з віддаленою розробкою та налагодженням вбудованих систем, можуть включати нестабільність з'єднання, обмежені ресурси вбудованих систем та затримки в зв'язку з віддаленістю.

Для вирішення перешкод при віддаленої розробки та налагодження включають в себе: використання надійних аппаратних налагоджувачів, вибір вбудованих систем з достатньою кількістю ресурсів для підтримки налагодження та використання технологій для зменшення затримок, таких як розширені протоколи для віддаленого налагодження. [3,6,8,9]

1.3 Протоколи та стандарти для віддаленого доступу

Розглянемо протоколи і стандарти, які використовуються для забезпечення віддаленого доступу до вбудованих систем.

Деякі з найважливіших компонентів вбудованої системи походять від

конкретні методології, які зазвичай називають стандартами. Стандарти диктують, як ці компоненти повинні бути розроблені, і які додаткові компоненти потрібні в системі, щоб забезпечити їх успішну інтеграцію та функціонування.

Стандарти, які суворо залежать від ринку, визначають функціональні можливості, які стосуються певної групи пов’язаних вбудованих систем, які мають подібні технічні характеристики або характеристики кінцевого користувача, зокрема:

1. Мережі та комунікації. Проміжні пристрої, що з’єднують мережеві кінцеві системи, такі пристрої, як концентратори, шлюзи, маршрутизатори та комутатори. Цей сегмент ринку також включає пристрої, що використовуються для аудіо/відео зв’язку, такі як

стільникові телефони (включаючи гібриди стільникового телефону/кишенькового комп’ютера), відеотелефони та банкомати.

2. Медична промисловість. Визначається як «...будь-який інструмент, апарат, пристрій, матеріал або інший виріб, незалежно від того, чи використовується окремо чи в комбінації, включаючи програмне забезпечення, необхідне для його належного програма, призначена виробником для використання людиною. Це включає в себе апарати для діалізу, інфузійні насоси, кардіомонітори, доставку ліків, протезування тощо.
3. Аерокосмічна промисловість і оборона. Системи, які впроваджуються в літаках або використовуються військовими, для керування польотом, «розумна» зброя та керування реактивним двигуном.
4. Автомобільна промисловість. Підсистеми, реалізовані в автомобілях, такі як розважальні центри, засоби керування двигуном, засоби безпеки, антиблокувальні гальма та прилади.
5. Побутова електроніка. Зазвичай включає пристрої, якими споживачі користуються в особистому житті, наприклад КПК (асистенти персональних даних), телевізори (аналогові та цифрові), побутову техніку (наприклад, мікрохвильові печі, посудомийні машини, пральні машини) та інтернет-прилади.
6. Промислова автоматизація та управління. «Розумні» роботизовані пристрої (розумні датчики, контролери руху, пристрої інтерфейсу «людина/машина», промислові перемикачі тощо), які використовуються переважно у виробничих галузях для виконання циклічного автоматизованого процесу.

У таблиці 1.3 наведено деякі поточні реальні стандарти та деякі цілі, які стоять за їх впровадженням. [5]

Таблиця 1.2 - Стандарти та цілі у вбудованих системах

Тип Стандарту		Стандарт	Ціль
Market Specific	Мережі та комунікації	TCP (Transmission Control Protocol)/IP (Internet Protocol)	Стек протоколів на основі RFC (запит на коментарі) 791(IP) і 793 (TCP), які визначають компоненти системного програмного забезпечення.
		PPP (Point-to-Point Protocol)	Компонент системного програмного забезпечення на основі RFC 1661, 1332 і 1334.
		IEEE (Institute of Electronics and Electrical Engineers) 802.3 Ethernet	Мережевий протокол, який визначає обладнання та систему програмні компоненти для локальних мереж (LAN)
		Cellular	Мережеві протоколи, реалізовані в стільникових телефонах, наприклад CDMA (множинський доступ з кодовим розділенням) і TDMA (множинний доступ з часовим розділенням), які зазвичай використовуються в США. TDMA є основою європ. Міжн. стандарту GSM цифрового стандарту UMTS

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
Market Specific	Медичні прилади	Medical Devices Directive	Європейська директива щодо медичних пристроїв – це стандарти для медичних пристроїв для держав-членів ЄС щодо аспектів безпеки та/або ефективності цих пристроїв. Пристрої, які обмінюються енергією з пацієнтом у терапев. спосіб або використ. для діагностики чи монітор. Захвор., належать до класу Iia. Пристрій, який з’єдн. з центральною системою кровообігу чи центр& нервовою системою, або містить лікарський засіб, тоді пристрій належить до класу III .
		IEEE1073 Medical Device Communications	Стандарти IEEE 1073 для зв'язку медичних пристроїв.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
Market Specific	Аерокосмічна промисловість і оборона	Department of Defense –JTA (Joint Technical Architecture)	Ініціативи Міністерства оборони, такі як Спільна технічна архітектура (JTA), забезпечують плавний потік інформації, необхідної для досягнення сумісності, що забезпечує оптимальну готовність. JTA було створено Міністерством оборони США, щоб визначити мінімальний набір стандартів інформаційних технологій, включаючи веб-стандарти, для досягнення військової сумісності.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
		Aerospace Industries Association of America, lnc	Ця служба стандартів включає національні аерокосмічні стандарти і метричні стандарти . Це обширна колекція, яка надає стандарти для компонентів, проектування та специфікації процесу для літаків, космічних апаратів, основних систем озброєнь і всі види електронних системи. Він також містить докумен. про закупівлю частин і компонентів високотехнологічних систем, включаючи кріпильні елементи, шланги високого тиску, фітинги, електричні з'єднувачі високої щільності, підшипники тощо.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
General Purpose	Безпека	Netscape IETF (Internet Engineering Task Force) SSL (Secure Socket Layer) 128-bit Encryption	SSL — це протокол безпеки, який забезпечує шифрування даних, автентифікацію сервера, цілісність повідомлень і необов’язкову автентифікацію клієнта для з’єднання TCP/IP і зазвичай інтегрований у браузери та веб-сервери.
		IEEE 802.10 Standards for Interoperable LAN/MAN Security (SILS)	Надає групу специфікацій на рівні апаратного та системного програмного забезпечення для реалізації безпеки в мережах.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
Market Specific	Автомобільна промисловість	ISO/TS 16949 – The Harmonized Standard for the Automotive Supply Chain	Спільно розроблено членами IATF (International Automotive Task Force) і формує вимоги до автомобільного виробництва та відповідних організацій з обслуговування запчастин. На основі автомобільних каталогів ISO 9001:2000, AVSQ (Італія), EAQF (Франція), QS-9000 (США) і VDA6.1 (Німеччина).
Market Specific	Побутова електроніка	JavaTV	Java TV Application Programming Interface (API) — це розширення платформи Java, яке надає доступ до функцій, унікальних для приймача цифрового телебачення, таких як: потокове аудіо-відео.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
		OSGi (Open Services Gateway Initiative)	Специфікація OSGi розроблена для вдосконалення всіх стандартів побутових мереж, таких як Bluetooth™, CAL, CEBus, Конвергенція, emNET, HAVi™, HomePNA™, HomePlug™, HomeRF™, технологія Jini™, LonWorks, UPnP, 802.11B і VESA. Інфраструктура та специфікації OSGi спрощують установку та роботу кількох служб на одному відкритому шлюзі служб (приставка, кабель або DSL). модем, ПК, мультимедійний шлюз)

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
		ISO/IEC 16500 DAVIC (Digital Audio Visual Council)	DAVIC — це галузевий стандарт наскрізної сумісності трансляції та інтерактивної цифрової аудіовізуальної інформації та мультимедійного зв’язку.
		HAVi (Home Audio Video Initiative)	HAVi надає стандарт домашньої мережі для бездоганної взаємодії цифрових аудіо- та відеоспоживчих пристроїв, дозволяючи всім аудіо- та відеопристроям у межах мережі, щоб взаємодіяти один з одним і дозволяти функціям одного чи кількох пристроїв керувати з іншого пристрою, незалежно від конфігурації мережі та виробника пристрою.

Продовження таблиці 1.2

Тип Стандарту		Стандарт	Ціль
Market Specific	Промисловий контроль	(EU) The Machinery Directive 98/37/EC	Директива ЄС для всіх машин, рухомих машин, а також машини для підйому і транспортування людей. Загалом обладнання, яке продається або викор. в ЄС, має відповідати застосовним обов'язковим основним вимогам охор. здоров'я та безпеки (EHSR). Більш. облад. , яке вважається менш небезпечним, може бути самостійно оцінено постачальником і мати можливість скласти технічний файл. 98/37/EC застосовується до збірки пов'язаних частин або компонентів із принаймні однією рухомою частиною

Кінець таблиці 1.2

Тип Стандарту		Стандарт	Ціль
		IEC (International Electrotechnical Commission 60204-1)	Застосовується до електричного та електронного обладнання промислових машин. Сприяє безпеці людей, які контактують з промисловими машинами, не лише від небезпек, пов’язаних з електрикою (таких як ураження електричним струмом і пожежа), але й від несправності самого електричного обладнання. Усуває небезпеки, пов’язані з машиною та її оточенням.

Таблиця 1.3 - Стандарти та цілі у вбудованих системах

1.4 Засоби програмування вбудованих систем

Вбудовані системи відзначаються великою різноманітністю технологій, архітектур, інструментальних засобів та мов програмування, які використовуються для їх розробки. Вбудовані програми часто розробляються в спеціалізованих інтегрованих середовищах розробки (IDE), які надають інструменти для написання коду, компіляції, відлагодження та випробувань.

- C та C++: Це дві найпоширеніші мови програмування для вбудованих систем. C мова використовується через свою швидкість та ефективність, а C++ додає об'єктно-орієнтовані особливості.
- Python: Python може бути використаний в деяких вбудованих системах, особливо коли швидкість виконання не є критичною. Python надає високорівневий синтаксис, який полегшує розробку. Python домінує в рейтингу мов програмування IEEE Spectrum. Він зайняв перше місце у випадках використання систем Web, Enterprise і Embedded.
- Assembler: Assembler може бути використаний для оптимізації коду для конкретних процесорів.
- Java та JavaScript: В деяких вбудованих системах використовуються Java та JavaScript, особливо коли вбудована система є частиною веб-орієнтованої архітектури.

Серед основних інструментів програмування для вбудованих систем можна відзначити:

- GCC: GNU Compiler Collection - набір компіляторів для різних мов, що використовуються в багатьох вбудованих системах.
- Eclipse: Це вільне та відкрите середовище розробки, яке широко використовується в розробці вбудованих систем. Воно має плагіни для підтримки різних мов та інструментів.
- PyCharm: Це кроссплатформенна інтегроване середовище розробки для мови програмування Python із можливістю безпосереднього запуску та налагодження будь-якого коду.
- Visual Studio Code: Це легкий кодовий редактор зі вбудованою підтримкою для розробки вбудованих систем через плагіни.
- PlatformIO: Це інструмент розробки вбудованих систем, який може бути використаний з Eclipse або Visual Studio Code.

Бібліотеки та фреймворки програмування для вбудованих на мові python можна відзначити:

- Adafruit_BBIO.GPIO: є бібліотекою для програмування вбудованих систем, зокрема для плати BeagleBone Black, за допомогою мови програмування Python. Ця бібліотека надає простий і зручний спосіб взаємодії з GPIO (загального призначення вводу-виводу) на платі BeagleBone Black. Вона дозволяє зчитувати стан вхідних пінів, управляти вихідними пінами, налаштовувати переривання та багато іншого.
- time: Модуль time є вбудованим в Python і надає функції для роботи зі часом. Він дозволяє отримувати поточний час, затримувати виконання програми на певний проміжок часу, вимірювати час виконання деяких операцій тощо. Цей модуль може бути корисним при розробці вбудованих систем, коли потрібно керувати часовими інтервалами або вимірювати продуктивність коду.
- PyBBIO: PyBBIO є фреймворком, спеціально розробленим для програмування вбудованих систем на платформі BeagleBone. Він надає простий та зрозумілий інтерфейс для взаємодії з GPIO, ADC (аналого-цифровий перетворювач), PWM (імпульсна ширина модуляції) та іншими функціями плати BeagleBone Black.[4,11]

1.5 Безпека при віддаленому доступі до вбудованих систем

Безпека є критичним аспектом при розробці вбудованих систем, особливо при віддаленому доступі. Розглянемо різні аспекти безпеки вбудованих систем при віддаленому доступі.

1.5.1 Складність вбудованих систем, які призводять до вразливостей

1. Складність програмного забезпечення

a. Рішення використовувати Linux у системах, які потребують вищих рівнів безпеки, є дискусійним.
   - Природа коду Linux із відкритим вихідним кодом вважається його сильною стороною, оскільки код отримує більше уваги та переглядається світовою спільнотою.
   - Linux також має свої недоліки, оскільки код постійно модифікується, що може призвести до появи вразливостей.

2. Апаратні складності
   a. Підключення до мережі вбудованих систем.
      - Традиційно, не будучи підключеними, вбудовані системи були захищені від ризиків, пов'язаних з Інтернетом.
      - В даний час зростає потреба у дистанційному управлінні пристроями та управлінні життєвим циклом пристроїв.
   b. Консолідації вбудованої системи. Зростає тенденція мати єдині, потужні вбудовані системи, які виконують кілька завдань або компонентів у порядку:
      - Краща взаємодія між ними.
      - Для економії коштів у деяких випадках.

Це зазвичай призводить до змішування завдань із високим рівнем безпеки із завданнями з низьким або некритичним рівнем безпеки.[7, 10]

1.5.2 Складності при розробці, які призводять до вразливостей

Незважаючи на зростаючу роль вбудованих систем, їхня безпека залишається слабкою, і переважна більшість спеціалістів із вбудованих систем мають лише елементарні знання щодо питань вбудованої безпеки. Багато професіоналів із вбудованих систем працювали з відносно невеликими базами коду та простими апаратними архітектурами, і їм важко впоратися з

тенденціями до більш складних конструкцій, підключення до мережі, консолідації системи та рішучих і добре фінансованих зловмисників.

Більше того, більшість спеціалістів із вбудованого програмного забезпечення не навчені мистецтву та науці проектування систем безпеки. Про безпеку слід думати пізніше, і її майже ніколи неможливо модернізувати.

Результатом є те, що вбудовані системи постачаються з уразливими місцями, які залишають критичну інфраструктуру відкритою. Професіонали з вбудованих систем повинні бути навчені найсучаснішому забезпеченню безпеки вбудованих систем, щоб вони могли інтегрувати та застосовувати ці практики для покращення безпеки.

1.5.3 Рішення безпеки, які запобігають вразливостям

1. Захист перед несанкціонованим доступом
    - Ізоляція компонентів
    - Застосування аутентифікації та авторизації
    - Використання сильних паролів
    - Оновлення та патчінг ПЗ
    - Тестування безпеки

Ізоляція компонентів — це основна політика безпеки вбудованих систем, необхідна для реалізації політик безпеки системи вищого рівня.

Наприклад, ядро операційної системи має бути ізольовано від розміщених у ньому додатків, щоб операційна система могла застосовувати політики контролю доступу до ресурсів вводу/виводу під її контролем. Без цієї ізоляції несправна або шкідлива програма може пошкодити ядро та перешкодити йому надавати будь-які інші служби безпеки. Ізоляція, в цьому контексті, відноситься як до часової, так і до просторової ізоляції; невиконана програма не повинна забороняти виконання інших програм або самого ядра.

Ще один гарний приклад композиційної сили політики ізоляції можна побачити в ефективному розгортанні криптографічних служб у системі. Такі програми, як IPsec або файлова система шифрування, які використовують

криптографію для реалізації своїх політик безпеки, залежать від ізоляції криптографічної підсистеми (особливо особистих ключів) від інших програм або підсистем. Без цієї ізоляції приватні ключі можуть бути розкриті, що зробить неможливим політику конфіденційності, передбачену захистом даних у русі (DIM) або даних у спокої (DAR).

2. Шифрування
   - Шифрування трафіку (TLS, SSL)
   - Застосування ключів і сертифікатів
   - Безпечне зберігання даних
3. Моніторинг та аудит
   - Збір і аналіз логів
   - Регулярні перевірки безпеки
   - Відстеження та реагування на інциденти безпеки
4. Захист від DDoS атак
   - Виявлення аномалій трафіку
   - Фільтрація пакетів
   - Міграція атак
5. Фізична безпека
   - Захист від несанкціонованого фізичного доступу
   - Моніторинг і забезпечення безпеки приміщень. [7,8,12,13]

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі були розглянуті основні теоретичні аспекти віддаленої розробки та налагодження вбудованих систем. Було визначено поняття вбудованих систем, Їх види - а саме характеристики для визначення категорії вбудованих систем та їх обмеження, а також типи програмних функцій, які процесор може виконувати у вбудованій системі.

Також був досліджений процес віддаленої розробки та налагодження вбудованих систем, перешкоди при віддаленій розроботці та налагодження та вирішення цих перешкод. Основні вирішення проблем віддаленої розробці є використання надійних аппаратних налагоджувачів, вибір вбудованих систем з достатньою кількістю ресурсів для підтримки налагодження та використання технологій для зменшення затримок, таких як розширені протоколи для віддаленого налагодження.

Велика увага була приділена протоколам та стандартам для віддаленого доступу до вбудованих систем. Були розглянуті різни основні групи стандартів та протоколів, які походять від

конкретні методології , таких як мережі та комунікації, медична промисловість, аерокосмічна промисловість і оборона, автомобільна промисловість, побутова електроніка, промислова автоматизація та управління.

Також були розглянуті різні засоби програмування вбудованих систем, зокрема мови програмування,основні інструменти програмування і фреймворки. Крім того, враховувалась безпека, складність вбудованих систем, складності при розробці, які призводять до вразливостей при віддаленому доступі до вбудованих систем і засоби її забезпечення.

РОЗДІЛ ІІ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ВІДДАЛЕНОЇ РОЗРОБКИ ТА НАЛАГОДЖЕННЯ ВБУДОВАНИХ СИСТЕМ

2.1 Огляд методів підключення вбудованих систем до локального комп'ютера

Є кілька варіантів, як можно підключити плату до локального комп’ютеру. У кожного із способів є свої переваги та недоліки, від яких і залежить вибір варіанта налаштування. Розглянемо ці варіанти.

2.1.1 Підключення через USB

1) Під’єднайте BeagleBone до комп’ютера за допомогою кабелю Mini USB, який постачається разом із платою.
2) Встановити драйвера

2.1.2 Підключення через SSH та USB

SSH дозволяє отримати доступ до наших файлів BeagleBone з віддаленої машини за допомогою команд терміналу.

- Windows. Якщо ви використовуєте Windows, вам потрібно завантажити та встановити безкоштовну програму під назвою PuTTY.
  1. Під’єднайте BeagleBone до комп’ютера за допомогою кабелю Mini USB.
  2. Відкрийте PuTTY.
  3. У діалоговому вікні налаштування PuTTY виберіть SSH. Та введіть 192.168.7.2 як хост. Номер порту має залишатися стандартним, тобто 22 порт.

- Mac OS X і Linux. У Mac OS X і Linux ви можете використовувати вікно терміналу за замовчуванням для встановлення зв'язку SSH, оскільки SSH є в усіх ОС Unix.
  1. Підключіть BeagleBone до комп'ютера за допомогою кабелю Mini USB.
  2. У вікні терміналу введіть sudo ssh debian@24.87.55.22

2.1.3 Підключення через SSH та Ethernet

Кабель Ethernet, підключений до BeagleBone, гарантує доступ до Інтернету. Цей доступ дуже зручний, оскільки може знадобитися інсталювати чи оновити програму або працювати над проектами, пов'язаними з Інтернетом.

Встановлення цього типу зв'язку також дає змогу отримати доступ до BeagleBone з будь-яких інших пристроїв, доки BeagleBone залишаєтесь підключеною до тієї самої мережі.

- Windows
  1. Увімкніть BeagleBone за допомогою кабелю Mini USB або джерела живлення 5 В постійного струму.
  2. Під'єднайте кабель Ethernet від маршрутизатора до BeagleBone.
  3. Встановіть з'єднання SSH через PuTTY.
- Mac OS X і Linux
  1. Увімкніть BeagleBone за допомогою кабелю Mini USB або джерела живлення 5 В постійного струму.
  2. Приєднайте кабель Ethernet від маршрутизатора до BeagleBone.
  3. Відкрити вікно терміналу. Введіть sudo ssh debian@24.87.55.22

2.1.4 Підключення вбудованої системи через послідовний порт USB

Послідовний порт — це спосіб передачі даних між BeagleBone Black та іншим пристроєм. Для встановлення послідовного зв'язку між комп'ютером і BeagleBone Black потрібен USB-TTL.

- Windows

1. Приєднайте USB-роз'єм кабелю TTL до комп'ютера.
2. Під'єднайте дроти до роз'ємів J1 на вашому BeagleBone Black:

   Чорний дріт до контакту 1

   Зелений дріт до контакту 4

   Білий дріт до контакту 5

3. Відкрийте PuTTY. У діалоговому вікні «Конфігурація PuTTY» виберіть «Послідовний(Serial)». Введіть назву послідовного порту вашого BeagleBone Black. Введіть 115200 у полі Швидкість.
4. Увімкніть BeagleBone Black за допомогою кабелю Mini USB.

- Mac OS X і Linux

1. Відкрийте нове вікно терміналу.Введіть.
2. Приєднайте USB-роз'єм кабелю TTL до комп'ютера.
3. Під'єднайте дроти до роз'ємів J1 на вашому BeagleBone Black:

   Чорний дріт до контакту 1

   Зелений дріт до контакту 4

   Білий дріт до контакту 5

4. Введіть ls /dev/tty*. І побачите новий пристрій, підключений до вашого комп'ютера.
5. Введіть sudo screen /dev/ttyUSB0 115200.
6. Увімкніть BeagleBone Black за допомогою кабелю Mini USB.

2.1.5 Переваги та недоліки підключень до вбудованої системи

Підключення через USB:

Переваги:

- Простота підключення.
- Одночасне живлення і передача даних за допомогою одного кабелю..
- Швидкість передачі даних: USB 2.0 пропонує швидкість передачі даних до 480 Mbps.

Недоліки:

- Обмеження по дальності: USB кабель має обмеження по довжині (5 метрів для USB 2.0).

Підключення через SSH через USB.

Переваги:

- Забезпечення безпеки: SSH надає зашифроване з'єднання, захищаючи ваші дані.
- Віддалене управління: SSH дозволяє віддалено управляти вбудованою системою.

Недоліки:

- Обмеження на швидкість: SSH через USB дотримується обмежень USB.
- Додаткові вимоги до налаштування: Вам потрібно налаштувати SSH на вбудованій системі і локальному комп'ютері.

Підключення через SSH через Ethernet.

Переваги:

- Велика швидкість передачі даних, що може досягати 1 Gbps.
- Можливість з'єднати пристрої на відстані до 100 метрів.

Недоліки:

- Потреба в додатковій налаштуванні мережі.

Підключення BeagleBone Black через послідовний порт USB.

Переваги:

- Надійний засіб комунікації, який широко використовується для відлагодження.
- Забезпечує пряме з'єднання з BeagleBone Black, обходячи мережеві служби.

Недоліки:

- Обмеження на швидкість: Послідовні порти, зазвичай, не підтримують високі швидкості передачі даних. [7, 11, 15]

2.2 Віддалений доступ до вбудованих систем через SSH

2.2.1. Визначення SSH

SSH (Secure Shell) - це протокол,який забезпечує захищене мережеве з'єднання і дозволяє віддалений доступ до вбудованих систем за допомогою командного рядка. SSH забезпечує шифрування трафіку та аутентифікацію, що робить його безпечним для використання в мережевих середовищах.

SSH це метод безпечного віддаленого входу з одного комп’ютера на інший. Він надає кілька альтернативних варіантів надійної автентифікації та захищає безпеку та цілісність зв’язку за допомогою надійного шифрування. Це безпечна альтернатива незахищеним протоколам входу (таким як telnet, rlogin) і незахищеним методам передачі файлів (таким як FTP).

2.2.2. Архітектура SSH

SSH має архітектуру клієнт/сервер див.рис. 2.1 Серверна програма SSH приймає або відхиляє вхідні підключення до свого головного комп’ютера. Потім користувачі запускають клієнтські програми SSH, як правило, на інших комп’ютерах, щоб надсилати запити до сервера SSH. Щоразу, коли комп’ютер

надсилає дані в мережу, SSH автоматично їх шифрує. Потім, коли дані досягають призначеного одержувача, SSH автоматично розшифровує їх.

Усі комунікації між клієнтами та серверами надійно зашифровані та захищені від модифікації.

Результатом є прозоре шифрування: користувачі можуть працювати в звичайному режимі, навіть не знаючи, що їхні повідомлення безпечно зашифровані в мережі. Крім того, SSH використовує сучасні безпечні алгоритми шифрування та є достатньо ефективним, щоб його можна було знайти в критично важливих програмах великих корпорацій.

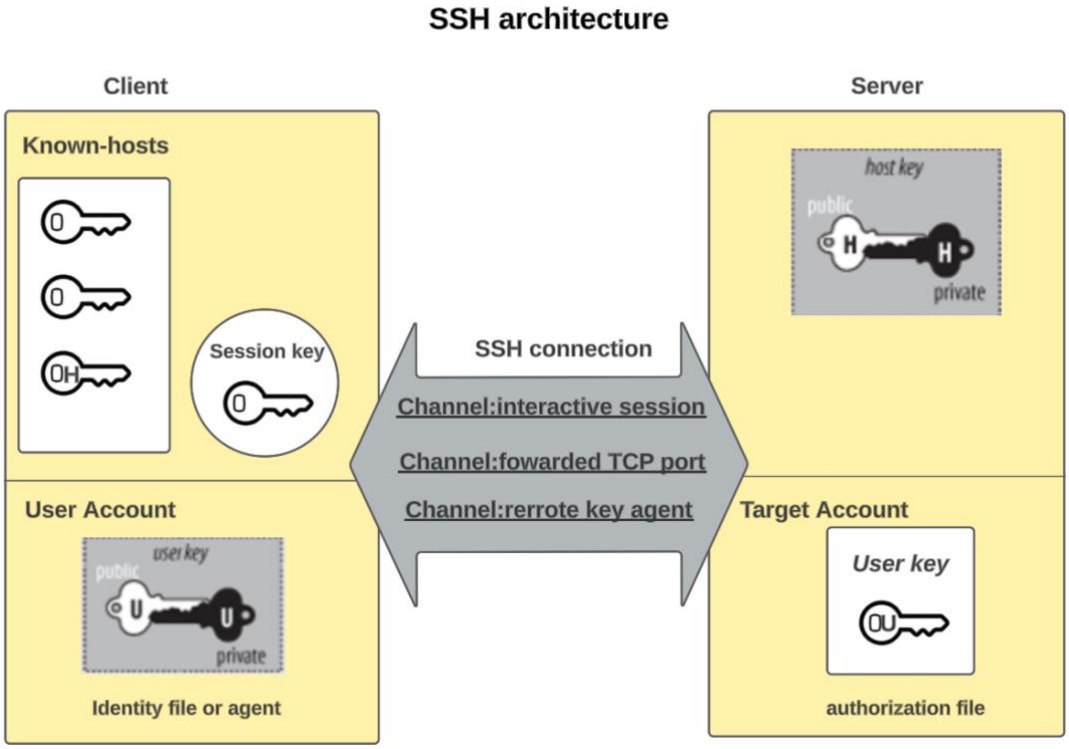


рис. 2.2.2.1 - Архітектура SSH

Особливості SSH:

1. Захищений протокол клієнт/сервер для шифрування та передачі даних через мережу.

   Протокол працює в моделі клієнт-сервер, що означає, що підключення встановлюється SSH-клієнтом, який підключається до SSH-сервера. Клієнт SSH керує процесом встановлення підключення та

використовує криптографію з відкритим ключем для перевірки ідентичності сервера SSH. Після етапу налаштування протокол SSH використовує надійне симетричне шифрування та алгоритми хешування для забезпечення конфіденційності та цілісності даних, якими обмінюються клієнт і сервер.

Є кілька варіантів, які можна використовувати для автентифікації користувача. Найпоширенішими з них є паролі та автентифікація за відкритим ключем.

2. Автентифікація (розпізнавання) користувачів за паролем, хостом або відкритим ключем, а також додаткова інтеграція з іншими популярними системами аутентифікації, такими як PAM, Kerberos, SecurID і PGP
3. Можливість додати безпеку незахищеним мережевим програмам, таким як Telnet, NNTP, VNC і багатьом іншим програмам і протоколам на основі TCP/IP
4. Майже повна прозорість для кінцевого користувача
5. Реалізації для більшості операційних систем
6. Метод автентифікації відкритого ключа

Метод автентифікації відкритого ключа полягає в тому, щоб мати пару криптографічних ключів – відкритий ключ і закритий ключ – і налаштувати відкритий ключ на сервері для авторизації доступу та надання доступу до сервера всім, хто має копію закритого ключа.

Основним використанням автентифікації на основі ключів є забезпечення безпечної автоматизації. Автоматичне захищене передавання файлів оболонки використовується для легкої інтеграції програм, а також для автоматизованих систем і керування конфігурацією.[15]

2.3 Використання VNC для віддаленої роботи з графічним інтерфейсом

2.3.1 Визначення VNC

VNC (Virtual Network Computing)- це протокол, що дозволяє віддалений доступ до графічного інтерфейсу вбудованої системи. Він передає зображення екрана з сервера на клієнтський комп'ютер та передає вхідні дані від клієнта до сервера. VNC дозволяє розробникам взаємодіяти з вбудованою системою, якби вони працювали безпосередньо на ній.

2.3.2. Архітектура VNC

VNC працює за моделлю клієнт/сервер. Компонент сервера встановлюється на віддаленому комп’ютері (той, яким ви хочете керувати), а програма перегляду VNC або клієнт встановлюється на пристрої, з якого ви хочете контролювати. Це може бути інший комп’ютер, планшет або мобільний телефон. Коли сервер і програма перегляду підключені, сервер передає копію екрана віддаленого комп’ютера програмі перегляду.

Сервер може надавати такі послуги, як дані або спільне використання ресурсів одному або кільком клієнтам. Таким чином один сервер може обслуговувати кілька клієнтів, а один клієнт може використовувати кілька серверів. Клієнт надсилає запит на сервер, який потім надсилає відповідь клієнту.

2.3.3 Визначення протоколу RFB

Remote Framebuffer, або RFB, — це протокол, який керує форматом даних, які передаються між клієнтом і сервером у системі VNC. Це те, що дозволяє клієнту віддалено переглядати інший комп’ютер і керувати ним. Він застосовний до всіх віконних програм і систем, що означає, що він працює на

таких платформах, як Windows, macOS, Linux та інших популярних операційних системах.

RFB починався як дуже простий протокол, але був розширений, щоб включити такі функції, як передача файлів, більш досконале стиснення та сильніші заходи безпеки в міру розвитку. Перехресна сумісність між клієнтами та серверами VNC стала можливою, оскільки вони можуть узгодити з'єднання, яке використовує найкращу версію RFB, а також параметри безпеки та стиснення, які підтримуються обома.

2.3.4. Огляд VNC

Декілька популярних реалізацій VNC - це все різні реалізації протоколу Virtual Network Computing (VNC), що використовуються для віддаленого доступу до графічних інтерфейсів користувача. Вони всі виконують в основному ті ж основні функції, але існують деякі важливі відмінності між ними:

1. RealVNC: Один з найбільш популярних клієнтів VNC, RealVNC це комерційний продукт, що містить безкоштовну версію для некомерційного використання та платні версії з додатковими функціями. RealVNC також включає підтримку захищеного з'єднання через шифрування і автентифікацію.[28]
2. TightVNC: Це вільнорозповсюджене та відкрите ПЗ, яке спрямоване на забезпечення покращеної продуктивності та безпеки. Він має декілька оптимізацій для покращення продуктивності, особливо при повільних з'єднаннях. Також TightVNC включає додаткову підтримку передачі файлів.[29]
3. UltraVNC: UltraVNC має додаткові функції, такі як: передача файлів, підтримка шифрування, підтримка "розмов" і підтримка з'єднання з кількома моніторами.[30]

4. TigerVNC: TigerVNC є вільним та відкритим варіантом клієнта VNC, що розробляється з акцентом на продуктивність та безпеку. Він базується на 4-й генерації VNC і включає підтримку шифрування, яке підтримується у всіх версіях протоколу. TigerVNC має багато вдосконалень щодо продуктивності порівняно з іншими клієнтами VNC, завдяки покращенням як в самому протоколі VNC, так і в драйверах відеокарт. TigerVNC є безкоштовним для використання в будь-якому середовищі, включаючи комерційні.[32]

RealVNC:

- Переваги: Висока надійність, підтримка шифрування даних, можливість налаштування рівня доступу та аутентифікації, підтримка багатьох платформ.
- Недоліки: Деякі функції доступні тільки в комерційних версіях, платний сервіс підтримки та оновлень.

TightVNC:

- Переваги: Добра продуктивність, висока швидкість передачі зображення, можливість налаштування стиснення даних, підтримка безкоштовна та відкрита програма.
- Недоліки: Недостатня підтримка шифрування, обмежені можливості налаштування доступу.

UltraVNC:

- Переваги: Додаткові функції, такі як перехоплення клавіатури та миші, передача файлів, додаткові інструменти для віддаленого керування.
- Недоліки: Потребує налаштування порту на маршрутизаторі, деякі проблеми зі сумісністю з деякими операційними системами.

TigerVNC:

- Переваги: Безкоштовний та відкритий, з акцентом на продуктивність та безпеку. Надає підтримку шифрування на всіх версіях протоколу. Висока продуктивність.
- Недоліки: Хоча і є потужним і гнучким рішенням, TigerVNC може вимагати більше технічних знань для налаштування та оптимізації.

2.4 JTAG та віддалене налагодження вбудованих систем

2.4.1. Визначення JTAG

JTAG (Joint Test Action Group) - це стандарт, який використовується для перевірки PCB(Printed Circuit Board) і програмування вбудованих систем. Він є основою для налагодження вбудованих систем на апаратному рівні.

Цей стандарт IEEE визначає, як перевіряється комп'ютерна схема для підтвердження її належної роботи після виробничого процесу. Випробування проводяться для перевірки паяних з'єднань на друкованих платах (Printed circuit board). Ця дія гарантує відсутність недоліків у з'єднаннях друкованої плати, а також відсутність проблем зі зв'язками та проводами зв'язку, які з'єднують контактні площадки інтегральної схеми з рамками контактів. JTAG надає тестерам огляд контактів з кожної контактної площадки інтегральної схеми, допомагаючи визначити будь-які несправності в друкованій платі.[9]

Інтерфейс JTAG можна додати до мікросхеми. Цей інтерфейс/порт може підключати зонд до чіпа, дозволяючи розробнику маніпулювати чіпом і його з'єднаннями з іншими чіпами. Розробники також можуть використовувати цей інтерфейс для копіювання мікропрограми в енергонезалежну пам'ять у межах електронний пристрій.[19]

2.4.2 Архітектура JTAG

JTAG складається з таких основних блоків:

1. Test Access Port (TAP): Це інтерфейс, який забезпечує зв'язок між JTAG-контролером та вбудованими компонентами на платі. TAP має декілька станів, таких як Test-Logic Reset, Run-Test/Idle, Shift-IR, Shift-DR та інші, і використовується для передачі команд та даних через JTAG.

Цей стандарт було розроблено, щоб забезпечити технологію тестування вузлів друкованих плат (Printed Circuit Board Assemblies) без потреби в фізичному доступі, необхідному для тестування.

TAP використовує наведені нижче сигнали для підтримки операції граничного сканування:

- TCK (Test Clock) – цей сигнал синхронізує роботу внутрішнього кінцевого автомата.
- TMS (Test Mode Select) – цей сигнал дискретизується на передньому фронті TCK для визначення наступного стану.
- TDI (Test Data In) – цей сигнал представляє дані, перенесені в тестову або програмну логіку пристрою. Він відбирається на передньому фронті TCK, коли внутрішній кінцевий автомат знаходиться в правильному стані.
- TDO (Test Data Out) – цей сигнал представляє дані, виведені з тестової або програмної логіки пристрою, і дійсний на спадному фронті TCK, коли внутрішній кінцевий автомат знаходиться в правильному стані.
- TRST (Test Reset) – це додатковий контакт, який, якщо доступний, може скинути кінцевий автомат контролера TAP.

2. Instruction Register (IR): Це регістр, який містить поточну інструкцію. Його вміст використовується контролером TAP, щоб вирішити, що

робити з отриманими сигналами. Найчастіше вміст регістру інструкцій визначатиме, до якого з регістрів даних слід передати сигнали.

Регістри даних – є три основні регістри даних: регістр граничного сканування (BSR), регістр BYPASS і регістр IDCODES. Інші регістри даних можуть бути присутніми, але вони не є обов'язковими як частина стандарту JTAG.

3. Boundary-Scan Register (BSR): Це регістр, який забезпечує можливість тестування зовнішніх з'єднань компонентів, які розміщені на платі.

Окремі біти або комірки цього регістра знаходяться на межі пристрою, між його функціональним ядром і контактами або кульками, за допомогою яких він підключений до плати. Boundary-Scan Register дозволяє виявляти дефекти в маршрутах провідників та перевіряти правильність підключення компонентів.

4. BYPASS – це однорозрядний регістр, який передає інформацію від TDI до TDO. Це дозволяє тестувати інші пристрої в ланцюзі з мінімальними витратами.
5. IDCODES – цей реєстр містить ідентифікаційний код і номер версії пристрою. Ця інформація дозволяє пов'язати пристрій із його файлом мови опису граничного сканування (BSDL). Файл містить деталі конфігурації граничного сканування для пристрою. [17,18]

2.4.3 Засоби розробки з використанням JTAG

Для розробки, налагодження та віддаленого керування вбудованими системами з використанням JTAG можна використовувати такі інструменти та програмне забезпечення:

1. OpenOCD (Open On-Chip Debugger): OpenOCD - це відкрите програмне забезпечення, яке надає можливість взаємодії з вбудованими

системами через JTAG. Воно підтримує різні контролери та мікроконтролери і має багато корисних функцій для налагодження та керування.

2. SEGGER J-Link: SEGGER J-Link - це комерційний JTAG-адаптер, який надає широкі можливості для налагодження та віддаленого керування вбудованими системами. Він підтримує різні мікроконтролери та має інтеграцію з популярними IDE (Integrated Development Environment).
3. Keil MDK (Microcontroller Development Kit): Keil MDK - це комплексний набір інструментів для розробки вбудованих систем, який включає підтримку JTAG для налагодження та керування мікроконтролерами.
4. Eclipse CDT (C/C++ Development Tools): Eclipse CDT - це безкоштовна інтегрована середовище розробки для мови C/C++, яке може бути розширене для підтримки JTAG-налагодження за допомогою відповідних плагінів.

2.5 HTTP-сервер для віддаленого моніторингу та управління

2.5.1 Визначення HTTP-серверу

HTTP (Hypertext Transfer Protocol): HTTP є протоколом передачі даних, який використовується для комунікації між веб-браузерами та веб-серверами. У контексті віддаленого доступу до вбудованих систем HTTP може використовуватися для створення веб-інтерфейсів, які дозволяють контролювати та моніторити вбудовані системи з використанням веб-браузерів.

HTTP Protocol використовується для обміну даними між клієнтом і сервером у формі документів на мові гіпертекстової розмітки (HTML). Ці документи включають зображення, текст, відеофайли та флеш-файли. HTTP працює на верхній частині всіх мережевих рівнів і відомий як прикладний

рівень і використовується для передачі всіх даних між підключеними мережевими пристроями.[16]

2.5.2 Огляд HTTP-серверу

HTTP-сервери можуть слугувати великою кількістю цілей, включаючи віддалений моніторинг та управління вбудованими системами. Популярних HTTP-серверів:

1. Apache HTTP Server: Це найпопулярніший веб-сервер в світі. Apache є відкритим ПЗ і підтримує широкий спектр модулів, які дозволяють йому слугувати різними функціями. Він має потужні можливості конфігурації і може бути налаштований на різні рівні безпеки.[21]
2. Django Server - це потужний і гнучкий веб-фреймворк, написаний на Python, який використовується для розробки складних веб-додатків. Django підтримує декілька протоколів HTTP і має вбудований HTTP-сервер, який можна використовувати для розробки та тестування.Django також має вбудовану підтримку аутентифікації та безпеки, яка може бути важливою для віддаленого управління.[20]
3. Nginx: Nginx є іншим дуже популярним веб-сервером, який також може слугувати як обернений проксі-сервер, завантажувач балансувальника навантаження і поштовий проксі-сервер. Він відомий високою продуктивністю та стабільністю.[22]
4. Lighttpd: Lighttpd, що вимовляється як "lighty", є веб-сервером, оптимізованим для високої продуктивності з низьким використанням пам'яті. Він має гнучку систему конфігурації, і його часто використовують для високонавантажених веб-серверів.[23]
5. Node.js: Хоча Node.js технічно не є веб-сервером, він може слугувати як такий за допомогою пакетів, як express.js. Node.js - це оточення

виконання JavaScript, що використовується для створення масштабованих мережевих програм. [24]

У кожного з HTTP-серверів є свої переваги та недоліки, розглянемо їх:

Apache HTTP Server:

Переваги:

- Багатофункціональність та підтримка різних модулів.
- Добре масштабується та забезпечує високу продуктивність.

Недоліки:

- Великий обсяг пам'яті та ресурсів, не найкращий варіант для обмежених пристроїв.
- Вимагає більше налаштувань для оптимальної роботи.

Django Server:

Переваги:

- Високорівневий фреймворк для розробки веб-додатків з підтримкою бази даних.
- Можливість використання Python для як серверного, так і для програмування вбудованих систем.
- Легкий у використанні.
- Забезпечує швидку розробку та добру масштабованість.

Недоліки:

- Менш гнучкий порівняно з іншими HTTP-серверами.
- Для вбудованих систем може бути зайвим навантаженням через свою високорівневість.

Nginx:

Переваги:

- Висока продуктивність.
- Низьке споживання ресурсів.
- Вбудована підтримка багатопотокового обслуговування з'єднань.
- Ефективно працює з великою кількістю одночасних з'єднань.

Недоліки:

- Не підтримує обробку динамічного контенту без додаткових модулів.
- Вимагає деякого досвіду для налаштування та розгортання.

Lighttpd:

Переваги:

- Легкий та швидкий HTTP-сервер з низьким споживанням ресурсів.
- Підтримка модулів та відносно проста конфігурація.
- Добре підходить для вбудованих систем з обмеженими ресурсами.

Недоліки:

- Обмежені можливості порівняно з іншими HTTP-серверами.
- Менш активна спільнота користувачів та розробників.

Node.js:

Переваги:

- Має високу продуктивність.
- Можливість використання JavaScript для як серверного, так і клієнтського коду.
- Широкий вибір модулів та бібліотек для розробки.

Недоліки:

- Вимагає розуміння асинхронного програмування та обробки подій.
- Може вимагати більшого обсягу пам'яті порівняно з іншими серверами.[6]

2.6 Хмарні сервіси для розробки та налагодження вбудованих систем

Останнім часом стає все популярнішим використання хмарних сервісів для розробки та налагодження вбудованих систем. Ці сервіси використовуються для віддаленого розгортання, керування та моніторингу вбудованих систем.

Інженерам не потрібно мати фізичний доступ до обладнання, оскільки вони можуть працювати з ним через хмару.

Хмарні сервіси дуже популярні в області вбудованих систем, особливо для розробки та налагодження. Ось деякі хмарні сервіси, які ви можете розглянути:

1. Amazon Web Services (AWS): AWS Greengrass дозволяє вбудованим пристроям виконувати обчислення на краю мережі, використовуючи AWS Lambda. AWS також пропонує IoT сервіси, такі як AWS IoT Core та AWS IoT Device Management, для підключення, керування та збору даних з вбудованих пристроїв.[25]
2. Microsoft Azure IoT Hub: Azure IoT Hub - це обслуговування хмари, яке дозволяє з'єднувати, моніторити та керувати мільярдами IoT активів. Він також включає Azure IoT Edge, який дозволяє розподіляти обчислення між краєм мережі та хмарою.[26]
3. Google Cloud IoT: Google Cloud IoT надає обслуговування для підключення, керування та обробки даних з вбудованих пристроїв. Він включає Cloud IoT Core для керування пристроями та Cloud Pub/Sub для обробки даних.[27]

Хмарні сервіси дозволяють швидко розгортати та масштабувати вбудовані системи, розробляти та налагоджувати їх віддалено та збирати та обробляти великі обсяги даних з пристроїв.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було проведено огляд різних існуючих рішень для віддаленої розробки та налагодження вбудованих систем. Були розглянуті такі засоби, як SSH, VNC, JTAG, HTTP-сервери та хмарні сервіси.

SSH (Secure Shell) є потужним протоколом для віддаленого доступу до вбудованих систем. Він забезпечує захищене з'єднання та можливість виконання різних команд і програм на віддаленому пристрої. Архітектура SSH включає клієнт-серверну модель та використовує криптографічні методи для забезпечення безпеки з'єднання.

VNC (Virtual Network Computing)- це протоколол, що дозволяє віддалений доступ до графічного інтерфейсу вбудованої системи. Він передає зображення екрана з сервера на клієнтський комп'ютер та передає вхідні дані від клієнта до сервера. Також ми розглянули декілька найпулярніших VNC, таких як: RealVNC, TightVNC, UltraVNC, та їх переваги та недоліки.

JTAG (Joint Test Action Group) є стандартом для віддаленого налагодження вбудованих систем. Цей стандарт використовує особливі порти та реєстри для доступу до внутрішніх компонентів пристрою і забезпечує можливість зчитування та запису даних, встановлення точок зупинки та стеження за станом системи. Також ми оглянули архітектуру JTAG та засоби її розробки , налагодження та віддаленого керування вбудованими системами.

HTTP-сервери використовуються для віддаленого моніторингу та управління вбудованими системами через протокол HTTP.

HTTP Protocol використовується для обміну даними між клієнтом і сервером у формі документів на мові гіпертекстової розмітки (HTML). HTTP-сервери надають можливість зчитувати та записувати дані, керувати параметрами системи та виконувати різні операції віддалено. Ми оглянули та порівняли кілька найпопулярніших HTTP-серверів, таких як Apache HTTP Server, Django Server, Nginx, Lighttpd, Node.js

Хмарні сервіси є потужними інструментами для розробки та налагодження вбудованих систем. Хмарні сервіси дозволяють швидко розгортати та масштабувати вбудовані системи, розробляти та налагоджувати їх віддалено та збирати та обробляти великі обсяги даних із пристроїв безпосередньо з хмари. Ми розглянули такі хмари як: Amazon Web Services, Microsoft Azure , Google Cloud.

Також були розглянуті методи підключення вбудованих систем до локального комп'ютера,їх переваги та недоліки. Такі як : підключення через USB, підключення через SSH та USB, підключення через SSH та Ethernet, підключення вбудованої системи через послідовний порт USB.

Кожен з цих засобів має свої переваги та обмеження, і вибір конкретного рішення залежить від потреб і вимог проекту. Існує багато причин для застосування методів віддаленої розробки та налагодження. Вбудовані системи часто розміщують у важкодоступних місцях або небезпечних середовищах. Ці інструменти відкривають широкі можливості для віддаленої розробки та налагодження, обновляти та тестувати вбудованих систем і сприяють збільшенню ефективності та зручності у цих процесах.

РОЗДІЛ ІІІ. ВИБІР АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ: ОБҐРУНТУВАННЯ ТА АНАЛИЗ

3.1 Вибір та обґрунтування використання специфічної вбудованої системи

3.1.1 Опис та технічні характеристики вбудованої системи BeagleBone Black

В якості основної апаратної платформи вбудованої системи для розробки та накладання було обрано материнську плату BeagleBone Black(див.рис. 3.1.1.1). BeagleBone Black є компактною вбудованою платформою з високим рівнем продуктивності, яка базується на 32-розрядному RISC-мікропроцесорі Sitara AM3358BZCZ100 Arm Cortex-A8 від Texas Instruments.

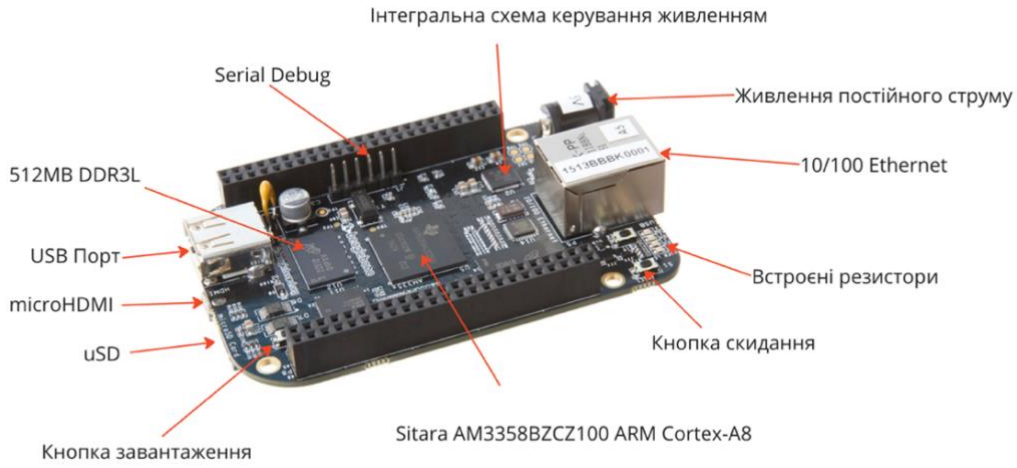


рис. 3.1.1.1 - Архітектура BeagleBone Black

Розглянемо технічні характеристики BeagleBone Black [30]:

Таблиця 3.1 -Технічні характеристики BeagleBone Black

Характеристика	BeagleBone Black
Процесор	Sitara AM3358BZCZ100 ARM Cortex-A8 32-Bit RISC, до 1GHz
Характеристика	BeagleBone Black
Швидкість процесора	До 1 ГГц
Вбудовані HDMI	Так, може прямо підключатися до телевізорів та моніторів
Оперативна пам'ять	512MB DDR3L 400MHz SDRAM
Вбудоване зберігання	4GB eMMC
Підтримка розширюючих плат	Так
Регулятор	TPS65217C PMIC та один додатковий LDO
Налагодження	Необов'язковий 20-контактний CTI JTAG
USB	Порт клієнта mini-AB USB 2.0, хост-порт USB 2.0 Type-A
Мережа	10/100 Ethernet

Продовження таблиці - 3.1

ПЗ	Linux Debian Distribution встановлений на eMMC
Акселератори	3D графічний акселератор, NEON floating-point акселератор
Характеристика	BeagleBone Black
Співпроцессори	2x200-MHz PRU, ARM Cortex-M3, SGX PowerVR
Сумісність з ПЗ	Debian, Android, Ubuntu, Cloud9 IDE на Node.js з бібліотекою BoneScript та багато іншого

3.1.2 Вибір та обґрунтування використання додаткового обладнання

Для підключення до плати BeagleBone Black було обрано USB-to-TTL Serial, miniUSB(для живлення) та cable та Ethernet-кабель.

Послідовний порт — це спосіб передачі даних між BeagleBone Black та іншим пристроєм. Для встановлення послідовного зв’язку між комп’ютером і BeagleBone Black потрібен послідовний кабель USB-to-TTL. USB-to-TTL Serial cable використовується для створення надійного з'єднання між розробницькою станцією та BeagleBone Black. Він забезпечує зручний та швидкий доступ до налагоджувальної консолі плати, дозволяючи виконувати різноманітні завдання по конфігурації та налагодженню системи.

Для налаштування під’єднання USB-to-TTL та BeagleBone Black треба:

1. Приєднуємо USB-роз’єм кабелю TTL до комп’ютера,
2. Введемо команду ls /dev/tty* для того, щоб побачити підключені пристрої

```
(base) regina@MacBook-Air-Regina ~ % ls /dev/tty*
/dev/tty                                /dev/ttysd
/dev/tty.Bluetooth-Incoming-Port        /dev/ttyse
/dev/tty.wlan-debug                     /dev/ttysf
/dev/ttyp0                              /dev/ttyt0
/dev/ttyp1                              /dev/ttyt1
/dev/ttyp2                              /dev/ttyt2
/dev/ttyp3                              /dev/ttyt3
/dev/ttyp4                              /dev/ttyt4
/dev/ttyp5                              /dev/ttyt5
/dev/ttyp6                              /dev/ttyt6
/dev/ttyp7                              /dev/ttyt7
/dev/ttyp8                              /dev/ttyt8
/dev/ttyp9                              /dev/ttyt9
/dev/ttypa                              /dev/ttyta
/dev/ttypb                              /dev/ttytb
/dev/ttypc                              /dev/ttytc
/dev/ttypd                              /dev/ttytd
/dev/ttype                              /dev/ttyte
/dev/ttypf                              /dev/ttytf
/dev/ttyq0                              /dev/ttyu0
/dev/ttyq1                              /dev/ttyu1
/dev/ttyq2                              /dev/ttyu2
/dev/ttyq3                              /dev/ttyu3
/dev/ttyq4                              /dev/ttyu4
/dev/ttyq5                              /dev/ttyu5
/dev/ttyq6                              /dev/ttyu6
/dev/ttyq7                              /dev/ttyu7
/dev/ttyq8                              /dev/ttyu8
/dev/ttyq9                              /dev/ttyu9
/dev/ttyqa                              /dev/ttyua
/dev/ttyqb                              /dev/ttyub
/dev/ttyqc                              /dev/ttyuc
/dev/ttyqd                              /dev/ttyud
/dev/ttyqe                              /dev/ttyue
/dev/ttyqf                              /dev/ttyuf
/dev/ttyr0                              /dev/ttyv0
```

рис. 3.1.2.1 - Вивід команди ls /dev/tty*

3. Під’єднуємо дроти до роз’ємів J1 на BeagleBone Black(див.рис. 3.1.2.2 і табл. 3.1.2.1):

Таблиця 3.2 - Приєднання дріту до номеру контакту USB-to-TTL

Дріт	Номер контакту
Чорний дріт	Pin 1
Зелений дріт	Pin 4

Кінець таблиці - 3.2

Дріт	Номер контакту
Білий дріт	Pin 5

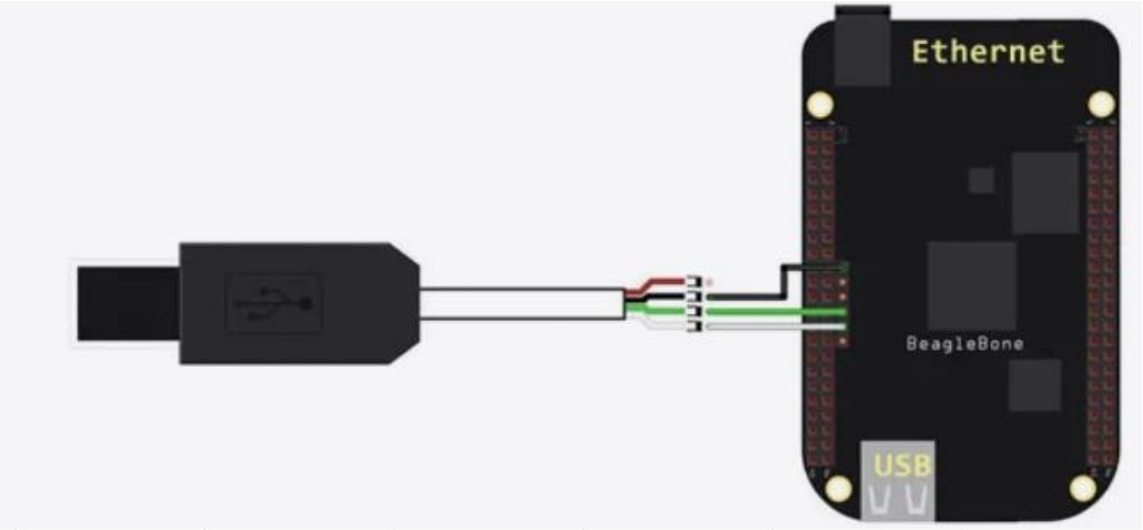


рис. 3.1.2.2 - Приєднання дротів USB-to-TTL.

4. Увімкнемо BeagleBone Black за допомогою кабелю Mini USB

5. Введимо ls /dev/tty*. Тепер ми можемо побачити новий пристрій, підключений до комп’ютера.

```
(base) regina@MacBook-Air-Regina ~ % ls /dev/tty*
/dev/tty                                /dev/ttysd
/dev/tty.Bluetooth-Incoming-Port        /dev/ttyse
/dev/tty.usbmodem2202SBB038586          /dev/ttysf
/dev/tty.wlan-debug                     /dev/ttyt0
/dev/ttyp0                              /dev/ttyt1
/dev/ttyp1                              /dev/ttyt2
/dev/ttyp2                              /dev/ttyt3
/dev/ttyp3                              /dev/ttyt4
/dev/ttyp4                              /dev/ttyt5
/dev/ttyp5                              /dev/ttyt6
```

рис. 3.1.2.3 - Вивід команди ls /dev/tty* після приєднання USB-to-TTL

6. Завантажимо screen для підключення до BeagleBone Black

```
(base) regina@MacBook-Air-Regina ~ % brew install screen
Running `brew update --auto-update`...
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
==> New Formulae
ansible@7              ddns-go                joshuto                spotify_player         votca
aws-amplify            fastgron               libecpint              swift-outdated         wzprof
bashate                git-credential-oauth   nexttrace              tern                   xbyak
==> New Casks
loupedeck                                                 motu-m-series

You have 4 outdated formulae and 1 outdated cask installed.
```

рис. 3.1.2.4 - Завантаження screen для підключення до BeagleBone

Команда screen є утилітою, яка дає можливість створювати термінали або вікна на одному терміналі. Вона дозволяє запустити декілька процесів терміналу одночасно та перемикатися між ними. Screen також зберігає стан терміналів, що дозволяє відновлювати їх після перезавантаження або відключення від сервера.

Використання команди screen дозволяє зручно взаємодіяти з пристроєм BeagleBone Black через послідовний порт USB та виконувати необхідні дії для розробки та налагодження вбудованих систем.

7. Вводимо команду “sudo screen /dev/tty.usbmodem2202SBB038586 115200” для підключення до BeagleBone Black

```
Debian GNU/Linux 10 beaglebone ttyGS0

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

beaglebone login: debian
Password:
Last login: Wed May 31 09:25:31 UTC 2023 from 10.8.0.6 on pts/1

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

рис. 3.1.2.5 - Вдале підключення до BeagleBone Black

Ethernet-кабель використовується для підключення плати до мережі Інтернет. Це дозволяє отримати доступ до віддалених ресурсів, оновити системне програмне забезпечення та встановити потрібні пакети. Крім того, з'єднання через Ethernet дає можливість використовувати протоколи, що базуються на TCP/IP, що є необхідним для реалізації деяких методів віддаленої розробки та налагодження.

3.2 Аналіз, вибір та обґрунтування використання VNC

3.2.1 Опис та характеристики VNC Xtigervnc

Xtigervnc - це варіація TigerVNC, що була розроблена спеціально для систем на базі Linux. Це високопродуктивна, і вже платформонезалежна система віддаленого доступу, яка використовує протокол RFB (Remote Frame Buffer) для візуалізації графічного інтерфейсу користувача на віддаленому комп'ютері.

Xtigervnc випускається під ліцензією GNU General Public License, що робить його вільним для використання та модифікації. А також Xtigervnc дає користувачам повний контроль над віддаленими системами, дозволяючи їм виконувати широкий спектр завдань від налагодження до розробки програмного забезпечення.[32]

Розглянемо характеристики Xtigervnc див.табл.3.2.1 :

Таблиця 3.3 - Характеристика Xtigervnc

Характеристика	VNC-Xtigervnc
Шифрування	Шифрування TLS
Множинні сесії	Так
Сумісність з RFB проток.	Повністю сумісний

Продовдення таблиці 3.3

Характеристика	VNC-Xtigervnc
Функції	Підтримка шифрування, розширений режим перегляду, автоматичне відтворення останніх сесій
Ліцензія	GNU GPL
Можливості персоналізації	Висока
Підтримка різних мережевих протоколів	Так
Віддалене адміністрування	Так
Керування декількома серверами	Так
Сумісність з HTTP-серверами	Так
Сумісність з різними браузерами	Так
Підтримка багатокористувацьких сесій	Так

Кінець - таблиці 3.3

Характеристика	VNC-Xtigervnc
Наявність вихідного коду	Відкритий
Наявність API	Так
Підтримка буфера обміну	Так
Підтримка управління сесіями	Розширена
Підтримка передачі звуку	Так
Підтримка передачі файлів	Так
Мінімальне навантаження на мережу	Високе
Можливість роботи через проксі-сервери	Так
Підтримка SSL/TLS шифрування	Так

3.3 Аналіз, вибір та обґрунтування використання HTTP-серверу

3.3.1 Визначення та переваги HTTP-сервера

Django - це високорівневий фреймворк для розробки веб-додатків на Python, який створює простоту розробки та прискорює процес виконання проекту. Django ідеально підходить для будь-якого проекту, де потрібно використовувати HTTP-сервер.

3.3.2 Обґрунтування використання Django для розробки HTTP-сервера

Переваги використання Django як HTTP-сервера включають:

- Повнота: Django надає повний набір функцій, які допоможуть вам з будь-яким веб-додатком.
- Безпека: Django допомагає розробникам уникнути багатьох поширених помилок безпеки, надаючи безпечну систему автентифікації користувачів та захищену систему обробки паролів.
- Скальованість: Django допомагає вам створювати додатки, які можуть легко масштабуватися для обслуговування великої кількості користувачів. [19]

Розглянемо характеристики Django HTTP-серверу:

Таблиця 3.4 - Характеристика Django HTTP-серверу

Характеристика	Django HTTP-сервер
Мова програмування	Python
Модель	MTV (Model-Template-View)
Безпека	Вбудовані механізми безпеки
Ліцензія	BSD-style

Продовження таблиці 3.4

Характеристика	Django HTTP-сервер
База даних	Підтримка багатьох систем управління бд
Середовище розробки	Вбудоване середовище розробки
Інтернаціоналізація	Повна підтримка багатомовності
Шаблонізатор	Дуже потужний і гнучкий шаблонізатор
Характеристика	Django HTTP-сервер
Форми	Потужна система форм
Кешування	Вбудовані механізми кешування
Сесії та кукі	Підтримка сесій та кукі
Розширюваність	Широкий вибір сторонніх додатків
REST API підтримка	Django Rest Framework дозволяє розробляти повноцінні REST API.
Панель Адміністратора	Django створює адміністративний інтерфейс для управління даними в моделях.
SQL	Django автоматично генерує SQL для моделей.

Кінець таблиці 3.4

Характеристика	Django HTTP-сервер
Django URL	Використовує регулярні вирази для визначення URL маршрутів.
Тестування	Вбудована підтримка тестування для створення модульних і системних тестів.
Автентифікація	Включає систему аутентифікації користувачів.
Підтримка версій HTTP	Підтримка HTTP/1.1 і HTTP/2.0.
Характеристика	Django HTTP-сервер
SSL/TLS підтримка	Підтримка SSL/TLS для безпечного обміну даними.
Підтримка WebSockets	Підтримка протоколу WebSockets для двостороннього зв'язку між клієнтом і сервером.
Застосунок для обробки статичних файлів	Підтримка статичних файлів, таких як CSS, JavaScript та зображення.

Таблиця 3.3.1 Характеристика Django HTTP-серверу

3.4 Оцінка та вибір хмарного сервісу

3.4.1 Визначення та переваги хмарних сервісів

Хмарні сервіси - це набір різноманітних сервісів, доступних через Інтернет. Вони забезпечують різні можливості, такі як обчислення, зберігання, бази даних, аналітика, інтелектуальні технології та ін. Хмарні сервіси мають багато переваг:

- Гнучкість: Хмарні сервіси дозволяють швидко масштабувати ресурси в залежності від потреб користувача.
- Економія витрат: Вам не потрібно витрачати кошти на покупку та обслуговування власних серверів.
- Доступність: Доступ до ресурсів хмари можливий з будь-якої точки з доступом до Інтернету.
- Резервне копіювання і відновлення даних: Більшість провайдерів хмарних сервісів забезпечують ці послуги, що знижує ризик втрати даних.

3.4.2 Вибір та обґрунтування використання AWS для віддаленої розробки та налагодження

Amazon Web Services (AWS) - це один з найбільш популярних хмарних сервісів, який пропонує широкий спектр продуктів та сервісів. AWS використовується для віддаленої розробки та налагодження з декількох причин:

- Велика кількість сервісів: AWS пропонує більше 200 повноцінних сервісів з центрів даних по всьому світу.
- Безпека: AWS надає високий рівень безпеки своїх сервісів.
- Надійність: AWS пропонує послуги з високою доступністю та довговічністю.[24]

Розглянемо характеристику Amazon Web Services у таблиці 3.4.1:

Таблиця 3.5 - Характеристика AWS

Характеристика	AWS
Розташування центрів даних	Більше 200 центрів даних по всьому світу
Бази даних	Підтримка різних видів баз даних (SQL, NoSQL, in-memory, key-value)
Обчислення	Різноманітність опцій обчислень, включаючи віртуальні машини, контейнери та безсерверні обчислення
Сервіси	Велика кількість сервісів, включаючи аналітику, AI, IoT, машинне навчання.
Надійність	Високий рівень доступності та довговічності, резервне копіювання та відновлення даних
Призначення	Для будь-яких типів застосувань, включаючи веб-хостінг, бізнес-аналітика, мобільні застосунки
Інтеграція з іншими сервісами	Високий рівень інтеграції з іншими сервісами AWS
(VPC)	Дозволяє створювати власну віртуальну мережу в межах AWS
Ідентифікація та доступ	Повний контроль над ідентифікацією користувачів та доступом до ресурсів AWS

Кінець таблиці 3.5

Зберігання	Підтримка різних видів зберігання (блочне, об'єктне, файлове), включаючи довготривале зберігання в Glacier
Географічний охоплення	Глобальний охоплення з регіональними центрами даних по всьому світу
Політика конфіденційності	AWS зобов'язується захищати конфіденційність і безпеку даних своїх клієнтів.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було обрано апаратне та програмне забезпечення, яке буде використано для реалізації проекту, зокрема вбудовану систему BeagleBone Black, протокол віддаленого доступу VNC Xtigervnc, HTTP-сервер Django та хмарний сервіс AWS. Було зроблено обґрунтування вибору цих ресурсів на основі аналізу їх технічних характеристик, можливостей, а також вимог до проекту.

Використання VNC для віддаленого доступу до системи було детально розглянуто. Переваги цього протоколу було проаналізовано в контексті вимог до проекту.

Було обрано HTTP-сервер Django для розробки веб-інтерфейсу системи. Обґрунтування цього вибору базувалось на потужних можливостях Django, а також його сумісності із Python, мовою програмування, яка буде використовуватися для розробки системи.

В кінці розділу було обрано хмарний сервіс AWS для віддаленої розробки та налагодження. Обґрунтування цього вибору було засноване на великому наборі сервісів, що надає AWS, його гнучкості та надійності, які забезпечують ефективність роботи над проектом.

РОЗДІЛ IV. НАЛАШТУВАННЯ І ВИКОРИСТАННЯ ВІДДАЛЕНОГО ДОСТУПУ ДО ВБУДОВАНИХ СИСТЕМ

4.1 Налаштування SSH для віддаленої роботи

4.1.1 Опис процесу налаштування SSH

SSH (Secure Shell) - це протокол,який забезпечує безпечне мережеве з'єднання і дозволяє віддалений доступ до вбудованих систем.

Щоб налаштувати SSH, спершу потрібно встановити SSH-сервер на віддаленій машині і SSH-клієнт на локальній машині. Розглянемо, як це зробити на різних операційних системах.

На macOS є вже вбудований SSH-клієнт та SSH-сервер, але його треба налаштувати, для цього в налаштуваннях вмикаємо "Віддалений доступ"

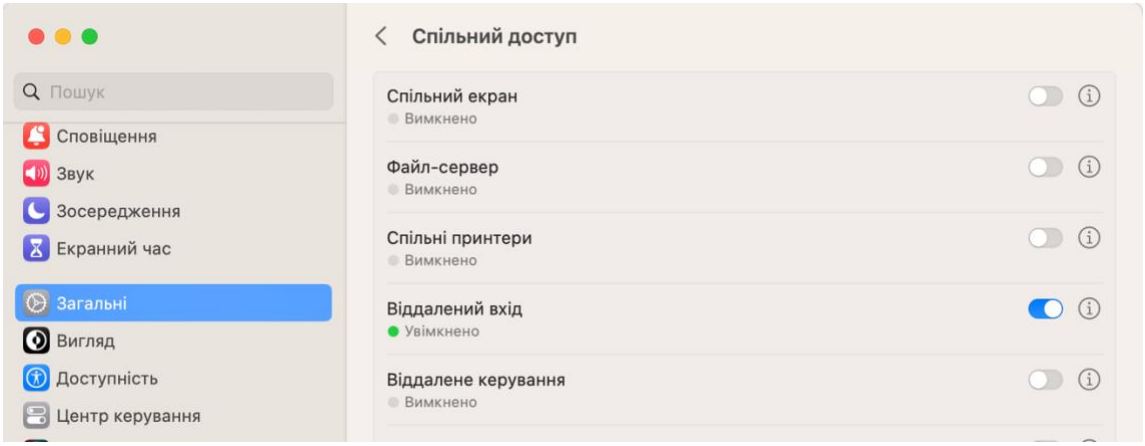


рис. 4.1.1.1 - Налаштування SSS на macOS

На Linux треба завантажити SSH пакет, для цього вводимо команду:

“sudo apt install openssh-client”

На Windows треба ввімкнути OpenSSH, для цього в "Панель керування" > "Програми" > "Увімкнути або вимкнути компоненти Windows" треба встановити прапорець на "OpenSSH Client" та "OpenSSH Server".

Або можно скачати программу Putty(вона підтримує протоколи мережевого з'єднання SSH, Telnet та SCP) для ssh-клієнту.

4.1.2 Демонстрація роботи SSH

Щоб до BeagleBone Black можно було віддалено підключатися, треба налаштувати підключення до Інтернету. Для цього ми використовуємо Ethernet-кабель, підключаючи його к порту Ethernet на BeagleBone Black та вільному порту на маршрутизаторі/комутаторі.

Щоб перевірити з'єднання з маршрутизатором і доступність Інтернету. Для цього знайдемо IP-адресу маршрутизатора, використовуючи команду:

“ip route | grep default”

```
debian@beaglebone:~$ ip route | grep default
default via 10.0.0.1 dev eth0
```

рис. 4.1.2.1 - Знаходження IP-адресу маршрутизатора

IP-адресу маршрутизатора - 10.0.0.1

Щоб перевірити з'єднання з маршрутизатором і доступність Інтернету, можна спілкуватися з ним за допомогою команди ping.

Виконаємо наступну команду, щоб перевірити чи правильно встановлене з'єднання з маршрутизатором:

"ping -c 5 10.0.0.1"

```
debian@beaglebone:~$ ping -c 5 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2.26 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.70 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.75 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=2.35 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=2.94 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 1.700/2.199/2.940/0.456 ms
```

рис. 4.1.2.1 - Перевірка з'єднання з маршрутизатором

Виконаємо наступну команду, щоб перевірити доступність Інтернету:

"ping -c 5 google.com"

```
debian@beaglebone:~$ ping -c 5 google.com
PING google.com(sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e)) 56 data by
tes
64 bytes from sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e): icmp_seq=1 t
tl=59 time=20.1 ms
64 bytes from sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e): icmp_seq=2 t
tl=59 time=19.6 ms
64 bytes from sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e): icmp_seq=3 t
tl=59 time=20.9 ms
64 bytes from sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e): icmp_seq=4 t
tl=59 time=18.0 ms
64 bytes from sea15s07-in-x0e.1e100.net (2607:f8b0:400a:800::200e): icmp_seq=5 t
tl=59 time=21.8 ms
```

рис. 4.1.2.2 - Перевірка доступу до інтернету

При підключенні до BeagleBone Black через SSH , ми будемо налаштовувати "Port Forwarding", необхідний для забезпечення доступу до SSH-сервера, який працює на BeagleBone Black, через Інтернет.

Port Forwarding (перенаправлення портів) - це техніка, яка дозволяє направляти зовнішній трафік на певний порт вашого маршрутизатора до конкретного пристрою в локальній мережі.

Без використання Port Forwarding, зовнішній трафік не зміг би досягти SSH-сервера, оскільки він призначений для локальної мережі.

Тобто, при налаштуванні, ми вказуємо, що будь-який трафік, який надходить на певний порт зовнішньої IP-адреси маршрутизатора, має бути перенаправлений на конкретний IP-адресу BeagleBone Black в локальній мережі.

Це дозволяє встановити з'єднання зі BeagleBone Black через Інтернет, використовуючи зовнішню IP-адресу маршрутизатора та встановлений порт перенаправлення. Без "Port Forwarding", не можна було б отримати доступ до BeagleBone Black зовні локальної мережі.

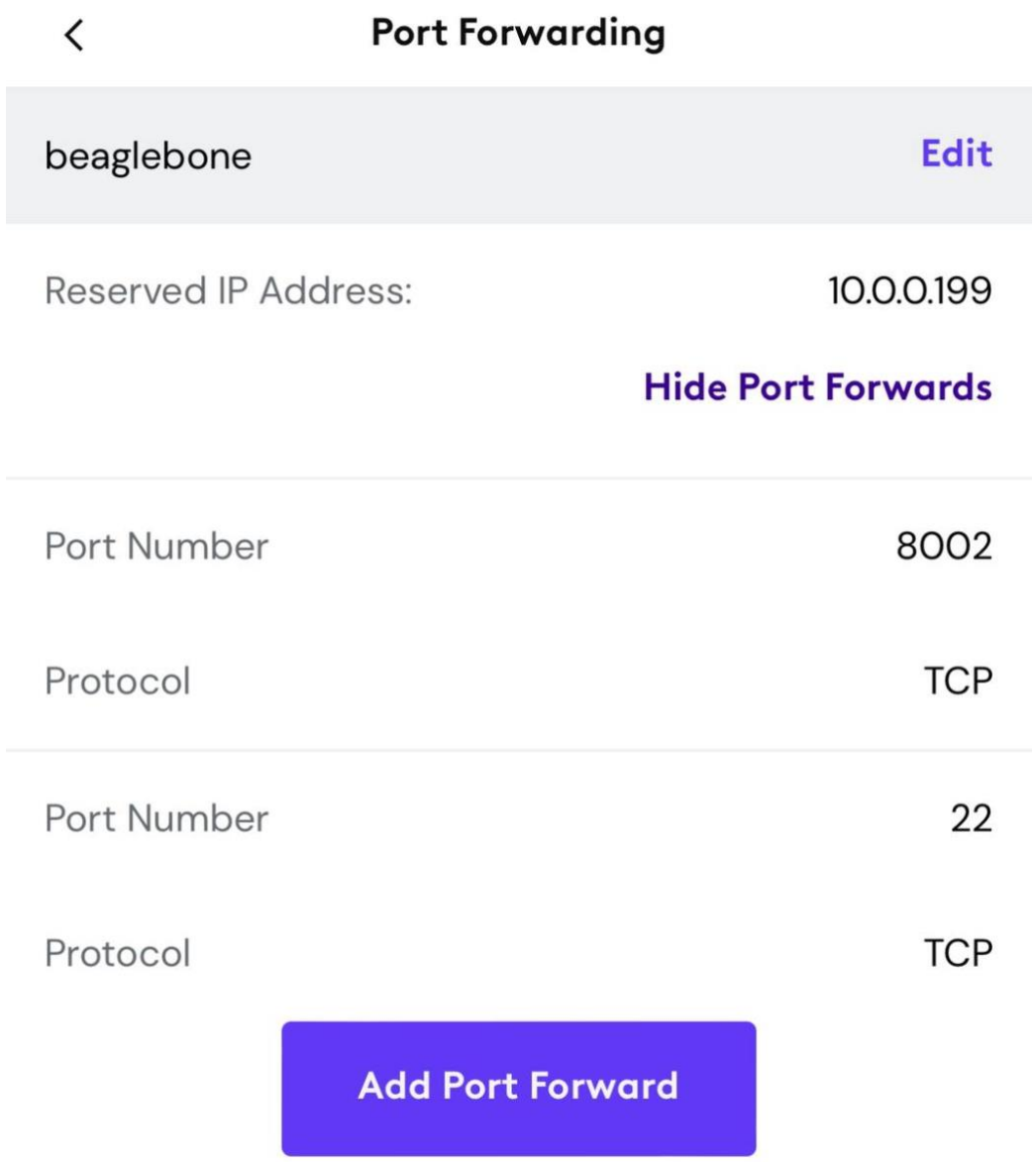


рис. 4.1.2.3 - Налаштування Port Forwarding

Для підключення по ssh до BeagleBone Black потрібно знати зовнішню IP-адресу маршрутизатора, для цього ми можемо виконати команду “curl ifconfig.me”.

```
debian@beaglebone:~$ curl ifconfig.me
24.87.55.22debian@beaglebone:~$
```

рис. 4.1.2.4 - Знайдення зовнішньої IP-адресу маршрутизатора

Після цих налаштувань, підключаємося по SSH до BeagleBone Black, використовуючи команду “ssh debian@24.87.55.22 -p 22”, де “24.87.55.22” - зовнішня IP-адресу маршрутизатора, “22” - порт, який ми налаштували у"Port Forwarding"

```
(base) regina@MacBook-Air-Regina ~ % ssh debian@24.87.55.22 -p 22
Debian GNU/Linux 10

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debian@24.87.55.22's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jun  1 15:19:11 2023
debian@beaglebone:~$
```

рис. 4.1.2.5 - Віддалений доступ до BeagleBone Black через SSH

4.2 Налаштування VNC для віддаленої роботи

VNC (Virtual Network Computing) - це технологія, що дозволяє віддалений доступ до графічного інтерфейсу вбудованої системи. Нам потрібно встановити VNC-сервер та VNC-клієнт.

4.2.1 Опис процесу налаштування VNC

Налаштування VNC Xtigervnc включає встановлення VNC сервера на віддаленому комп'ютері - на BeagleBone Black та встановлення VNC клієнта на локальному комп'ютері, а потім налаштування з'єднання між цими двома комп'ютерами.

Для налаштування VNC сервера завантажимо Xtigervnc пакет, для цього вводимо команду: “sudo apt-get install tigervnc-standalone-server”

Встановлюємо пароль для доступу до сервера VNC, використовуючи команду "vncpasswd":

```
debian@beaglebone:~$ vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? y
Password:
Verify:
```

рис. 4.2.1.1 - Встановлення паролю до VNC Xtigervnc

Запускаємо сервер VNC, використовуючи команду "vncserver -localhost no", щоб можно було ідключатися не тільки з localhost:

```
debian@beaglebone:~$ vncserver -localhost no

New 'beaglebone.localdomain:1 (debian)' desktop at :1 on machine beaglebone.localdomain

Starting applications specified in /etc/X11/Xvnc-session
Log file is /home/debian/.vnc/beaglebone.localdomain:1.log

Use xtigervncviewer -SecurityTypes VncAuth,TLSVnc -passwd /home/debian/.vnc/passwd beaglebone.localdomain:1 to connect to the VNC ser
ver.
```

рис. 4.2.1.1 - Успішно запущений сервер VNC Xtigervnc

Це запустить сервер VNC на платі BeagleBone Black. За замовчуванням, сервер буде слухатися на порту 5900 + "номер дисплея". Номер дисплея за замовчуванням - 1, тому сервер VNC слухатиме на порту 5901. Нам потрібен цей порт, щоб з VNC -клієнту підключитеся до сервера VNC.

Перевіримо чи сервер VNC працює коректно і прослуховує порт 5901, для цього введемо команду "sudo netstat -tuln | grep 5901":

```
debian@beaglebone:~$ netstat -tuln | grep 5901
tcp        0      0 0.0.0.0:5901            0.0.0.0:*               LISTEN
tcp6       0      0 :::5901                 :::*                    LISTEN
```

рис. 4.2.1.2 - Перевірка прослуховування порта 5901

Це значить, що сервер VNC прослуховує порт 5901 і що він приймає не лише локальні підключення, і ми можемо підключитися до нього з клієнту, вказавши цей порт.

Налаштовуємо "Port Forwarding", щоб встановити з'єднання зі BeagleBone Black через Інтернет, використовуючи зовнішню IP-адресу маршрутизатора та встановлений порт перенаправлення 5901.

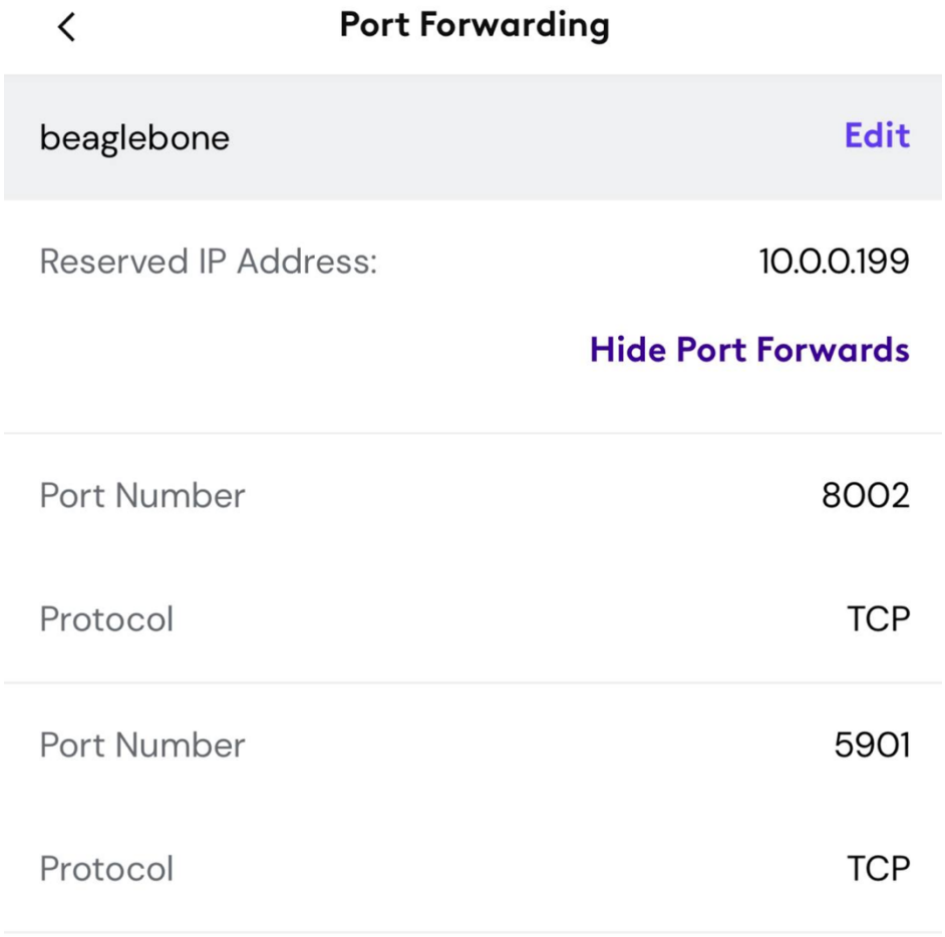


рис. 4.2.1.3 - Налаштування Port Forwarding для порта 5901

4.2.2 Демонстрація роботи VNC

Для підключення до VNC Xtigervnc серверу, Для macOS та Windows треба завантажити TigerVNC з офіційного сайту та встановити та запустити TigerVNC Viewer. А для Linux треба встановіть VNC-клієнт, за допомогою команди "sudo apt install tigervnc-viewer" та запустити TigerVNC Viewer з терміналу, виконавши команду "xtigervncviewer".

Перевіряємо список активних сесій сервера VNC:

```
debian@beaglebone:~$ vncserver -list

TigerVNC server sessions:

X DISPLAY #     RFB PORT #      PROCESS ID
:1              5901            2517
```

рис. 4.2.2.1 - Список активних сесій сервера VNC

Для підключення з клієнту будемо використовувати такі данні: “24.87.55.22” - зовнішня IP-адресу маршрутизатора, “5901” - порт, який ми налаштували у "Port Forwarding"

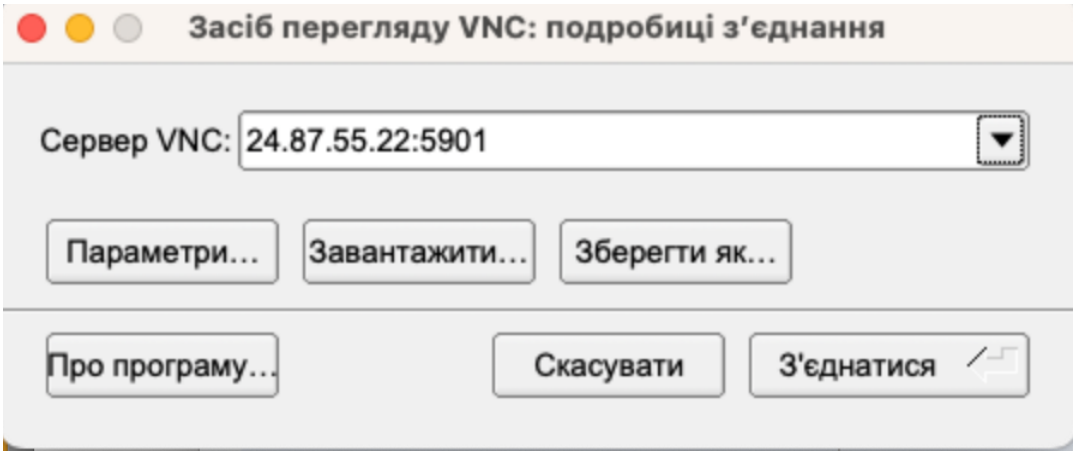


рис. 4.2.2.2 - Підключення клієнту VNC до VNC серверу

Вводимо пароль, який встановленний на VNC сервері:

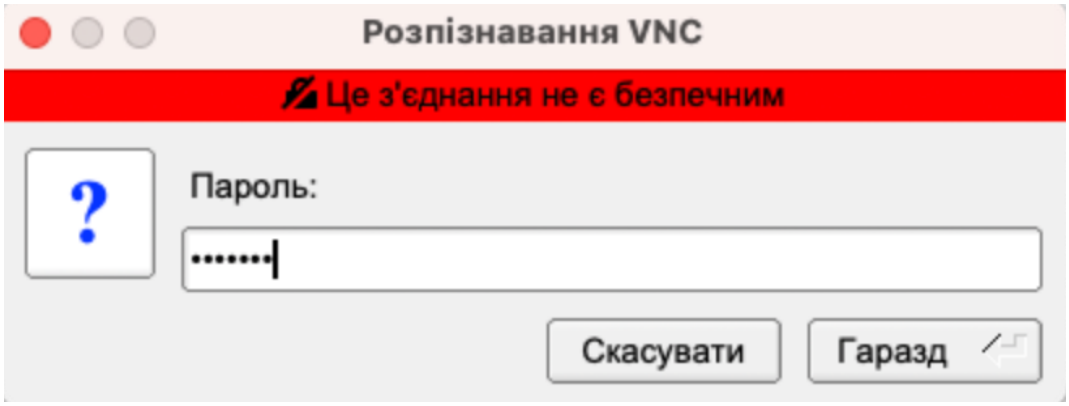


рис. 4.2.2.3 - Підключення клієнту VNC до VNC серверу

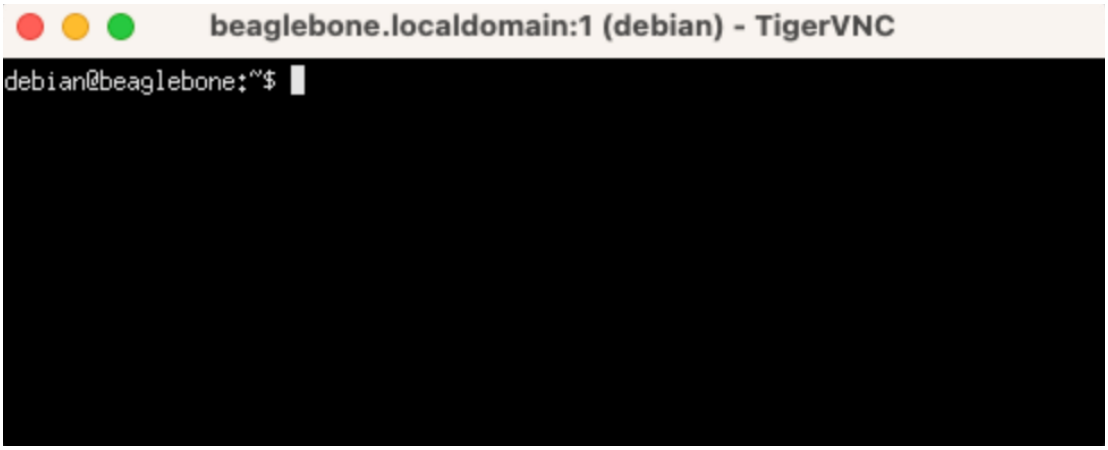


рис. 4.2.2.4 - Вдале підключення клієнту VNC до VNC серверу

Перевіримо з’єднання на клієнті, для цього використовуємо команду "netstat -tnpa | grep 5901" відображає поточні мережеві з'єднання, пов'язані з портом 5901.

```
debian@beaglebone:~$ netstat -tnpa | grep 5901
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:5901            0.0.0.0:*               LISTEN      2517/Xtigervnc
tcp        0      0 10.0.0.199:5901         24.87.55.22:50104       ESTABLISHED 2517/Xtigervnc
tcp6       0      0 :::5901                 :::*                    LISTEN      2517/Xtigervnc
```

рис. 4.2.2.4 - Поточні мережеві з'єднання, після підключення клієнту

Строка ‘tcp 0 0 10.0.0.199:5901 24.87.55.22:50104 ESTABLISHED 2517/Xtigervnc” - показує активне з'єднання порту 5901 з віддаленим клієнтом 24.87.55.22 порту 50104. [32]

4.3 Налаштування HTTP-серверу для віддаленої роботи

4.3.1 Опис процесу налаштування HTTP-серверу

Для налаштування HTTP-серверу будемо використовувати фреймворк Django. Створимо основну структуру проекту Django в PyCharm.

Для цього виконаємо такі команди:

1) Встановлюємо Django: "pip install django"
2) Створюємо новий проект Django: "django-admin startproject myproject", переходимо в цю папку "cd myproject"
3) Створюємо новий додаток Django: "python manage.py startapp myapp"

Встановимо Python та Django на Debian, для цього використовуємо команди: "sudo apt-get update", "sudo apt-get install python3-pip", "sudo pip3 install django"

```
debian@beaglebone:~/BeadleBonePython$ sudo apt-get update
[sudo] password for debian:
Sorry, try again.
[sudo] password for debian:
Sorry, try again.
[sudo] password for debian:
Hit:1 http://deb.debian.org/debian buster InRelease
Get:2 http://deb.debian.org/debian buster-updates InRelease [56.6 kB]
Hit:3 http://deb.debian.org/debian-security buster/updates InRelease
Hit:4 http://repos.rcn-ee.com/debian buster InRelease
Fetched 56.6 kB in 4s (12.7 kB/s)
Reading package lists... Done
debian@beaglebone:~/BeadleBonePython$ sudo apt-get install python3-venv
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython3.7 libpython3.7-dev libpython3.7-minimal libpython3.7-stdlib
  python3.7 python3.7-dev python3.7-minimal python3.7-venv
Suggested packages:
  python3.7-doc binfmt-support
The following NEW packages will be installed:
  python3-venv python3.7-venv
The following packages will be upgraded:
  libpython3.7 libpython3.7-dev libpython3.7-minimal libpython3.7-stdlib
  python3.7 python3.7-dev python3.7-minimal
7 upgraded, 2 newly installed, 0 to remove and 195 not upgraded.
Need to get 53.0 MB of archives.
After this operation, 49.2 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-dev armhf 3.7.3-2+deb10u4 [513 kB]
Get:2 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-dev armhf 3.7.3-2+deb10u4 [47.2 MB]
Get:3 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7 armhf 3.7.3-2+deb10u4 [1,281 kB]
Get:4 http://deb.debian.org/debian-security buster/updates/main armhf python3.7 armhf 3.7.3-2+deb10u4 [330 kB]
Get:5 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-stdlib armhf 3.7.3-2+deb10u4 [1,659 kB]
Get:6 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-minimal armhf 3.7.3-2+deb10u4 [1,465 kB]
Get:7 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-minimal armhf 3.7.3-2+deb10u4 [582 kB]
Get:8 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-venv armhf 3.7.3-2+deb10u4 [6,136 B]
Get:9 http://deb.debian.org/debian buster/main armhf python3-venv armhf 3.7.3-1 [1,180 B]
Fetched 53.0 MB in 6s (8,443 kB/s)
(Reading database ... 74064 files and directories currently installed.)
Preparing to unpack .../0-python3.7-dev_3.7.3-2+deb10u4_armhf.deb ...
Unpacking python3.7-dev (3.7.3-2+deb10u4) over (3.7.3-2+deb10u1) ...
Preparing to unpack .../1-libpython3.7-dev_3.7.3-2+deb10u4_armhf.deb ...
Unpacking libpython3.7-dev:armhf (3.7.3-2+deb10u4) over (3.7.3-2+deb10u1) ...
```

рис. 4.3.1.2 - Вивід команди "sudo apt-get update"

```
debian@beaglebone:~/BeadleBonePython$ sudo apt-get update
[sudo] password for debian:
Sorry, try again.
[sudo] password for debian:
Sorry, try again.
[sudo] password for debian:
Hit:1 http://deb.debian.org/debian buster InRelease
Get:2 http://deb.debian.org/debian buster-updates InRelease [56.6 kB]
Hit:3 http://deb.debian.org/debian-security buster/updates InRelease
Hit:4 http://repos.rcn-ee.com/debian buster InRelease
Fetched 56.6 kB in 4s (12.7 kB/s)
Reading package lists... Done
debian@beaglebone:~/BeadleBonePython$ sudo apt-get install python3-venv
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython3.7 libpython3.7-dev libpython3.7-minimal libpython3.7-stdlib
  python3.7 python3.7-dev python3.7-minimal python3.7-venv
Suggested packages:
  python3.7-doc binfmt-support
The following NEW packages will be installed:
  python3-venv python3.7-venv
The following packages will be upgraded:
  libpython3.7 libpython3.7-dev libpython3.7-minimal libpython3.7-stdlib
  python3.7 python3.7-dev python3.7-minimal
7 upgraded, 2 newly installed, 0 to remove and 195 not upgraded.
Need to get 53.0 MB of archives.
After this operation, 49.2 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-dev armhf 3.7.3-2+deb10u4 [513 kB]
Get:2 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-dev armhf 3.7.3-2+deb10u4 [47.2 MB]
Get:3 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7 armhf 3.7.3-2+deb10u4 [1,281 kB]
Get:4 http://deb.debian.org/debian-security buster/updates/main armhf python3.7 armhf 3.7.3-2+deb10u4 [330 kB]
Get:5 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-stdlib armhf 3.7.3-2+deb10u4 [1,659 kB]
Get:6 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-minimal armhf 3.7.3-2+deb10u4 [1,465 kB]
Get:7 http://deb.debian.org/debian-security buster/updates/main armhf libpython3.7-minimal armhf 3.7.3-2+deb10u4 [582 kB]
Get:8 http://deb.debian.org/debian-security buster/updates/main armhf python3.7-venv armhf 3.7.3-2+deb10u4 [6,136 B]
Get:9 http://deb.debian.org/debian buster/main armhf python3-venv armhf 3.7.3-1 [1,180 B]
Fetched 53.0 MB in 6s (8,443 kB/s)
(Reading database ... 74064 files and directories currently installed.)
Preparing to unpack .../0-python3.7-dev_3.7.3-2+deb10u4_armhf.deb ...
Unpacking python3.7-dev (3.7.3-2+deb10u4) over (3.7.3-2+deb10u1) ...
Preparing to unpack .../1-libpython3.7-dev_3.7.3-2+deb10u4_armhf.deb ...
Unpacking libpython3.7-dev:armhf (3.7.3-2+deb10u4) over (3.7.3-2+deb10u1) ...
```

рис. 4.3.1.3 - Вивід команди "sudo apt-get install python3-pip"

```
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting django
  Downloading https://files.pythonhosted.org/packages/57/12/da22535f809b8c06c8d5
8eaf236ec8683ffd4e1dc4eced175b174e6446fa/Django-3.2.18-py3-none-any.whl (7.9MB)
    100% |████████████████████████████████| 7.9MB 20kB/s
```

рис. 4.3.1.4 - Вивід команди "sudo pip3 install django"

Використовуючи команду “scp” - це команда, яка використовується для копіювання файлів між хостами по мережі, використовуючи протокол SSH для передачі даних, забезпечуючи тим самим приватність та безпеку передаваних даних. Перенесемо проект Django з MacOs Pycharm на Debian, використовуємо команду: “scp -r /Users/regina/PycharmProjects/pythonProject13/myproject debian@24.87.55.22:/home/debian/BeadleBonePython”

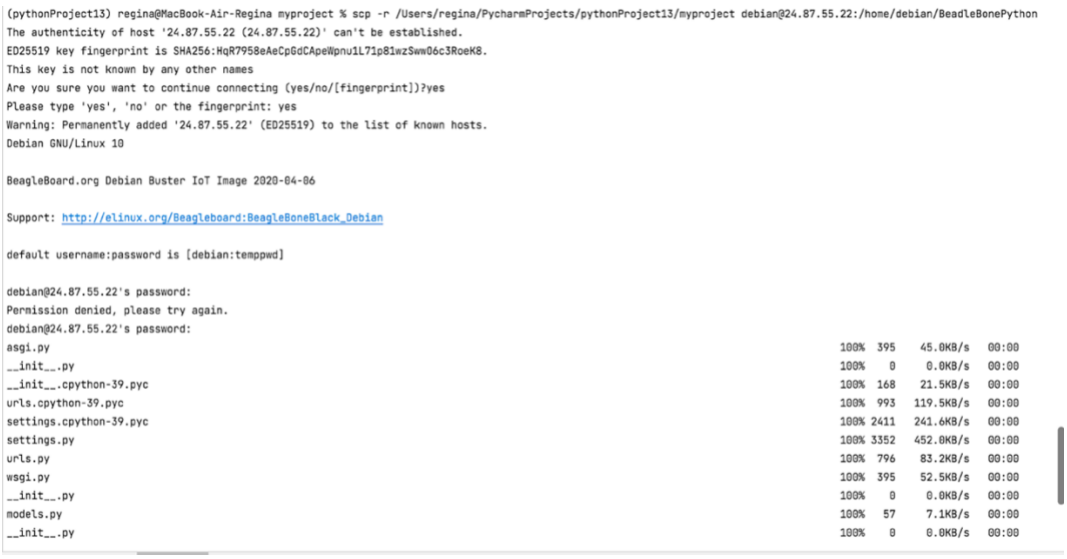


рис. 4.3.1.5 Безпечний перенос Django проекту на Debian

На BeagleBone Black створюємо віртуальне середовище, використовуючи команду: "python3 -m venv my_venv" і активуємо віртуальне середовище за допомогою команди "source my_venv/bin/activate"

```
debian@beaglebone:~/BeadleBonePython$ python3 -m venv BeadleBone
debian@beaglebone:~/BeadleBonePython$ ls /home/debian/BeadleBonePython
BeadleBone  myproject  my_venv
debian@beaglebone:~/BeadleBonePython$ source /home/debian/BeadleBonePython/Beadl
eBone/bin/activate
(BeadleBone) debian@beaglebone:~/BeadleBonePython$
```

рис. 4.3.6. - Створення і активування віртуального середовища

Встановлюємо Adafruit_BBIO.GPIO: є бібліотекою для програмування вбудованих систем, зокрема для плати BeagleBone Black, за допомогою мови програмування Python, яка надає простий і зручний спосіб взаємодії з GPIO на платі BeagleBone Black. Використовуємо команду “sudo pip3 install Adafruit_BBIO”

```
(BeadleBone) debian@beaglebone:~/BeadleBonePython$ pip install Adafruit_BBIO

Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting Adafruit_BBIO
  Downloading https://www.piwheels.org/simple/adafruit-bbio/Adafruit_BBIO-1.2.0-cp37-cp37m-lin
ux_armv7l.whl (298kB)
    100% |████████████████████████████████| 307kB 445kB/s
Installing collected packages: Adafruit-BBIO
Successfully installed Adafruit-BBIO-1.2.0
```

рис. 4.3.1.7 Вивід команди sudo pip3 install Adafruit_BBIO

path('', views.index, name='index') - URL-шаблон для кореневого шляху. При зверненні до кореневого шляху буде викликатись функція index з модуля views.

path('toggle/<str:led>/', views.toggle_led, name='toggle_led') - URL-шаблон для шляху "toggle/<str:led>/". При зверненні до цього шляху буде викликатись функція toggle_led з модуля views. Значення <str:led> буде передано у функцію подання як аргумент led.

path('toggle_all/', views.toggle_all, name='toggle_all') - URL-шаблон для шляху "toggle_all/". При зверненні до цього шляху буде викликатись функція toggle_all з модуля views.

path('running_leds/', views.running_leds, name='running_leds') - URL-шаблон для шляху "running_leds/". При зверненні до цього шляху буде викликатись функція running_leds з модуля views.

path('random_leds_blink/',views.random_leds_blink,name='random_leds_blink') - URL-шаблон для шляху "random_leds_blink/". При зверненні до цього шляху буде викликатись функція random_leds_blink з модуля views. Налаштування urls.py див.рис. 4.3.1.8

```
  GNU nano 3.2                          urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('toggle/<str:led>/', views.toggle_led, name='toggle_led'),
    path('toggle_all/', views.toggle_all, name='toggle_all'),
    path('running_leds/', views.running_leds, name='running_leds'),
    path('random_leds_blink/', views.random_leds_blink, name='random_leds_blink$
]
```

рис. 4.3.1.8 Налаштування urls.py

Файл views.py Django є модулем Python, який містить функції, які визначають логіку обробки HTTP-запитів і повертають відповідні HTTP-відповіді. У файлі views.py визначено такі функції:

led_pins – це словник, який співвідносить імена світлодіодів (LED1, LED2, LED3, LED4) з відповідними пінами GPIO (USR0, USR1, USR2, USR3). Потім у циклі відбувається налаштування GPIO пінів як виходи.

```
from django.shortcuts import render
from django.http import JsonResponse
import random
import time
import Adafruit_BBIO.GPIO as GPIO

led_pins = {"LED1": "USR0", "LED2": "USR1", "LED3": "USR2", "LED4": "USR3"}

for pin in led_pins.values():
    GPIO.setup(pin, GPIO.OUT)
```

рис. 4.3.1.9 Налаштування views.py

toggle_led(request, led): Функція перемикає стан світлодіода led і повертає JSON-відповідь з інформацією про результат операції.

```
#Перемикає стан світлодіода
def toggle_led(request, led):
    set_led_status(led, not get_led_status(led))
    response_data = {
        "result": "success",
        "message": f"LED {led} {'ON' if get_led_status(led) else 'OFF'}",
    }
    return JsonResponse(response_data)
```

рис. 4.3.1.10 Налаштування views.py функція toggle_led()

get_led_status(led): Функція повертає поточний стан LED.

set_led_status(led, status): Функція встановлює стан світлодіода led значення status.

index(request): Функція обробляє запити до головної сторінки та повертає шаблон index.html.

```
#Повертає поточний стан світлодіода
def get_led_status(led):
    return GPIO.input(led_pins[led])

#Встановлює стан світлодіода
def set_led_status(led, status):
    GPIO.output(led_pins[led], status)


def index(request):
    return render(request, 'index.html')
```

рис. 4.3.1.11 Налаштування views.py функції get_led_status(), set_led_status(), index()

toggle_led(request, led): Функція перемикає стан світлодіода led і повертає JSON-відповідь з інформацією про результат операції.

Функція toggle_led(request, led) приймає параметр led, який свідчить про конкретний світлодіод, стан якого потрібно змінити. За допомогою функцій get_led_status(led) та set_led_status(led, status) відбувається отримання поточного стану світлодіода та встановлення протилежного стану. Потім створюється JSON-відповідь, що містить результат виконання операції та повідомлення, що вказує на стан світлодіоду.

```
#Перемикає стан світлодіода
def toggle_led(request, led):
    set_led_status(led, not get_led_status(led))
    response_data = {
        "result": "success",
        "message": f"LED {led} {'ON' if get_led_status(led) else 'OFF'}",
    }
    return JsonResponse(response_data)
```

рис. 4.3.1.12 Налаштування views.py функції toggle_led()

toggle_all(request): Функція перемикає стан всіх світлодіодів і повертає JSON-відповідь з інформацією про результат операції.

Функція toggle_all(request) перемикає стан всіх світлодіодів. Спочатку визначається поточний стан одного із світлодіодів (current_state). Потім визначається новий стан, протилежний поточному (new_state). За допомогою функції GPIO.output(pin, new_state) змінюється стан усіх світлодіодів. Створюється JSON-відповідь з результатом виконання операції та повідомленням, що вказує на новий стан усіх світлодіодів.

```
#Перемикає стан всіх світлодіодів
def toggle_all(request):
    current_state = GPIO.input(led_pins["LED1"])

    new_state = not current_state
    for pin in led_pins.values():
        GPIO.output(pin, new_state)

    response_data = {
        "result": "success",
        "message": f"All LEDs {'ON' if new_state else 'OFF'}",
    }
    return JsonResponse(response_data)
```

рис. 4.3.1.13 Налаштування views.py функції toggle_all()

Функція running_leds(request) є логікою послідовного включення і вимкнення всіх світлодіодів з певною затримкою між ними.

Якщо метод запиту дорівнює 'POST', функція отримує значення затримки (delay) із параметрів запиту POST.

Цикл проходить по всіх світлодіодах зі списку led_pins. На кожній ітерації світлодіод встановлюється стан "Включено" (set_led_status(led, True)), потім відбувається пауза (time.sleep(delay)), після чого світлодіод встановлюється стан "Вимкнено" (set_led_status(led, False)).

Після завершення циклу функція повертає JSON-відповідь із полем success, що вказує на успішне виконання операції.

Якщо метод запиту не є 'POST', функція повертає JSON-відповідь з помилкою та повідомленням "Invalid request method".

```
#Вмикає кожен світлодіод послідовно з певною затримкою між ними.
def running_leds(request):
    if request.method == 'POST':
        delay = float(request.POST.get('delay', 0.1))
        for led in led_pins:
            set_led_status(led, True)
            time.sleep(delay)
            set_led_status(led, False)
        return JsonResponse({'success': True})
    else:
        return JsonResponse({'error': 'Invalid request method'})
```

рис. 4.3.1.14 Налаштування views.py функції running_leds()

Функція toggle_led_without_request(led) є логікою перемикання стану світлодіоду без необхідності отримання запиту від клієнта.

Вона приймає аргумент led, який представляє назву світлодіода зі словника led_pins. Функція використовує set_led_status(led, status) для перемикання світлодіодного стану.

Всередині функції викликається функція get_led_status(led), щоб отримати поточний стан світлодіода, а потім за допомогою оператора не інвертується цей

стан. Результат передається в set_led_status(led, status) зміни стану світлодіода на протилежне.

Таким чином, функція toggle_led_without_request дозволяє перемикати стан зазначеного світлодіода між увімкненим та вимкненим без необхідності отримання запиту від клієнта.

```
#Вмикає і вимикає випадковий світлодіод декілька разів з певною затримкою між
def toggle_led_without_request(led):
    set_led_status(led, not get_led_status(led))
```

рис. 4.3.1.15 Налаштування views.py функції toggle_led_without_request()

Функція random_leds_blink(request) є логікою миготіння випадковими світлодіодами з певною затримкою між ними.

Якщо метод запиту дорівнює 'POST', функція отримує значення затримки (delay) та кількість миготінь (blink_count) із параметрів POST-запиту.

У циклі повторюється blink_count разів. На кожній ітерації вибирається випадковий світлодіод зі списку доступних світлодіодів (led_pins.keys()), використовуючи random.choice(). Потім викликається функція toggle_led_without_request(led) для перемикання стану вибраного світлодіода.

Після цього відбувається пауза (time.sleep(delay)), щоб створити затримку між блиманнями світлодіода.

Наприкінці функція повертає JSON-відповідь із полем success, що вказує на успішне виконання операції.

```
def random_leds_blink(request):
    if request.method == 'POST':
        delay = float(request.POST.get('delay', 0.1))
        blink_count = int(request.POST.get('count', 10))

        for _ in range(blink_count):
            led = random.choice(list(led_pins.keys()))
            toggle_led_without_request(led)
            time.sleep(delay)
            toggle_led_without_request(led)

        return JsonResponse({'success': True})

else:
    return JsonResponse({'error': 'Invalid request method'})
```

рис. 4.3.1.16 Налаштування views.py функції random_leds_blink()

Код у файлі script.js містить кілька функцій, які використовують метод fetch для надсилання AJAX-запитів до сервера та отримання даних у форматі JSON.

toggleLed(led) - функція для перемикання стану світлодіода із зазначеним ім'ям (led). Вона відправляє GET-запит на шлях /toggle/<str:led>/ і виводить отримані дані в консоль. До цієї функції ми звертаємося 4 рази у файлі index.html onclick="toggleLed('LED1'), onclick="toggleLed('LED2'), onclick= "toggleLed('LED3'), onclick="toggleLed('LED4').

```
function toggleLed(led) {
    fetch(`/toggle/${led}/`)
        .then(response => response.json())
        .then(data => console.log(data));
}
```

рис. 4.3.1.17 Налаштування sqript.js функції toggleLed(led)

```
<div id = "led1" class="ledClass">
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED1')">Увімкнути/вимкн$
</div>
```

рис. 4.3.1.18 Налаштування index.html виклик toggleLed(‘LED1’)

```
    <button class="buttonClass" onclick="toggleLed('LED1')">Увімкнути/вимкн$
</div>
<div id="led2" class="ledClass">
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED2')">Увімкнути/вимкн$
</div>
<div id = "led3" class="ledClass" >
```

рис. 4.3.1.19 Налаштування index.html виклик toggleLed(‘LED2’)

```
<div id = "led3" class="ledClass" >
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M.01 0h24v24h-24z" fill="none"></path>
        <path d="M19.01 3h-14c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED3')">Увімкнути/вимкн$
</div>
<div id="led4" class="ledClass" >
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED4')">Увімкнути/вимкн$
</div>
```

рис. 4.3.1.20 Налаштування index.html виклик toggleLed(‘LED3’)

```
<div id = "led3" class="ledClass" >
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M.01 0h24v24h-24z" fill="none"></path>
        <path d="M19.01 3h-14c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED3')">Увімкнути/вимкн$
</div>
<div id="led4" class="ledClass" >
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2$
    </svg>
    <button class="buttonClass" onclick="toggleLed('LED4')">Увімкнути/вимкн$
</div>
```

рис. 4.3.1.21 Налаштування index.html виклик toggleLed(‘LED4’)

toggleAll() - функція перемикання стану всіх світлодіодів. Вона надсилає GET-запит на шлях /toggle_all/ і виводить отримані дані в консоль. До цієї функції ми звертаємось у файлі index.html onclick="toggleAll().

```
function toggleAll() {
    fetch('/toggle_all/')
        .then(response => response.json())
        .then(data => console.log(data));
}
```

рис. 4.3.1.22 Налаштування sqript.js функції toggleAll()

runningLeds() - функція для запуску вогнів, що біжать, на світлодіодах. Вона відправляє POST-запит на шлях /running_leds/ із затримкою (delay) у запиті. Також у заголовках запиту передається CSRF-токен (X-CSRFToken), який зазвичай використовується для захисту підробки запитів. Отримані дані виводяться у консоль. До цієї функції ми звертаємось у файлі index.html onclick="runningLeds()".

```
function runningLeds() {
    fetch('/running_leds/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': window.csrfToken
        },
        body: JSON.stringify({ delay: 0.1 })
    })
        .then(response => response.json())
        .then(data => console.log(data));
}
```

рис. 4.3.1.23 Налаштування sqript.js функції runningLeds()

randomLedsBlink() - функція для випадкового миготіння світлодіодами. Вона відправляє POST-запит на шлях /random_leds_blink/ із затримкою (delay) та кількістю миготінь (count) у тілі запиту. Аналогічно, передається CSRF-токен у заголовках запиту. Отримані дані виводяться у консоль. До цієї функції ми звертаємось у файлі index.html onclick="randomLedsBlink()".

```
function randomLedsBlink() {
    fetch('/random_leds_blink/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': window.csrfToken
        },
        body: JSON.stringify({ delay: 0.1, count: 10 })
    })
        .then(response => response.json())
        .then(data => console.log(data));
}
```

рис. 4.3.1.24 Налаштування sqript.js функції randomLedsBlink()

```
</div>
<div class="allLeds" >
    <button class="allLedsButton" onclick="toggleAll()">Увімкнути всі $
    <button class="allLedsButton" onclick="runningLeds()">Бігаючі резистори</bu$
    <button class="allLedsButton" onclick="randomLedsBlink()">Випадкове ми$
</div>
```

рис. 4.3.1.25 Налаштування index.html виклик toggleAll(), runningLeds(), randomLedsBlink()

X-CSRFToken - це HTTP-заголовок, який використовується для передачі CSRF-токена при виконанні AJAX-запитів за допомогою JavaScript. CSRF-токен є унікальним значенням, що генерується сервером і пов'язане з користувальницькою сесією. Він використовується для перевірки справжності запиту та захисту від атак, у яких зловмисник намагається виконати дії від імені авторизованого користувача.[19]

```
{% csrf_token %}
<script>window.csrfToken = "{{ csrf_token }}";</script>
<script src="{% static 'script.js' %}"></script>
ead>
```

рис. 4.3.1.26 Налаштування X-CSRFToken в index.html

4.3.2 Демонстрація роботи HTTP-серверу

Після застосування всіх налаштувань та налаштувань ми запускаємо сервер, викликаючи команду "python3 manage.py runserver 0.0.0.0:8002". Висновок команди дає таку інформацію:

"System check identified no issues (0 silenced)." - Системна перевірка не виявила проблем.

"Starting development server на http://0.0.0.0:8002/" - сервер розробки запущений за адресою http://0.0.0.0:8002/.

Потім слідують записи про запити (GET/POST) та коди стану HTTP-відповідей. В даному випадку видно запити на перемикання стану світлодіодів та їх коди стану (200 - успішно).

```
debian@beaglebone:~/BeadleBonePython/myproject$ python3 manage.py runserver 0.0.0.0:8002
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, conten
ttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 02, 2023 - 00:37:39
Django version 3.2.18, using settings 'myproject.settings'
Starting development server at http://0.0.0.0:8002/
Quit the server with CONTROL-C.
[02/Jun/2023 00:37:39] "GET / HTTP/1.1" 200 2991
[02/Jun/2023 00:37:39] "GET /static/script.js HTTP/1.1" 304 0
[02/Jun/2023 00:37:39] "GET /static/styles.css HTTP/1.1" 304 0
[02/Jun/2023 00:37:41] "GET /toggle/LED1/ HTTP/1.1" 200 47
[02/Jun/2023 00:37:42] "GET /toggle/LED1/ HTTP/1.1" 200 48
[02/Jun/2023 00:37:42] "GET /toggle/LED2/ HTTP/1.1" 200 47
[02/Jun/2023 00:37:43] "GET /toggle/LED2/ HTTP/1.1" 200 48
[02/Jun/2023 00:37:44] "GET /toggle/LED3/ HTTP/1.1" 200 47
[02/Jun/2023 00:37:44] "GET /toggle/LED4/ HTTP/1.1" 200 47
[02/Jun/2023 00:37:45] "GET /toggle_all/ HTTP/1.1" 200 47
[02/Jun/2023 00:37:47] "POST /running_leds/ HTTP/1.1" 200 17
[02/Jun/2023 00:37:48] "POST /random_leds_blink/ HTTP/1.1" 200 17
```

рис. 4.3.2.1 Запуск http-серверу

На скріншоті представлена адаптивна веб-сторінка для керування станами світлодіодів. З такими кнопками як, увімкнути/вимкнути окремий світлодіод, увімкнути/вимкнути усі світлодіоди, кнопка "Бігаючі вогоньки", а також кнопка "Випадкове миготіння". Дані про відповіді сервера виконання запиту так само відображаються в інструментах розробника.

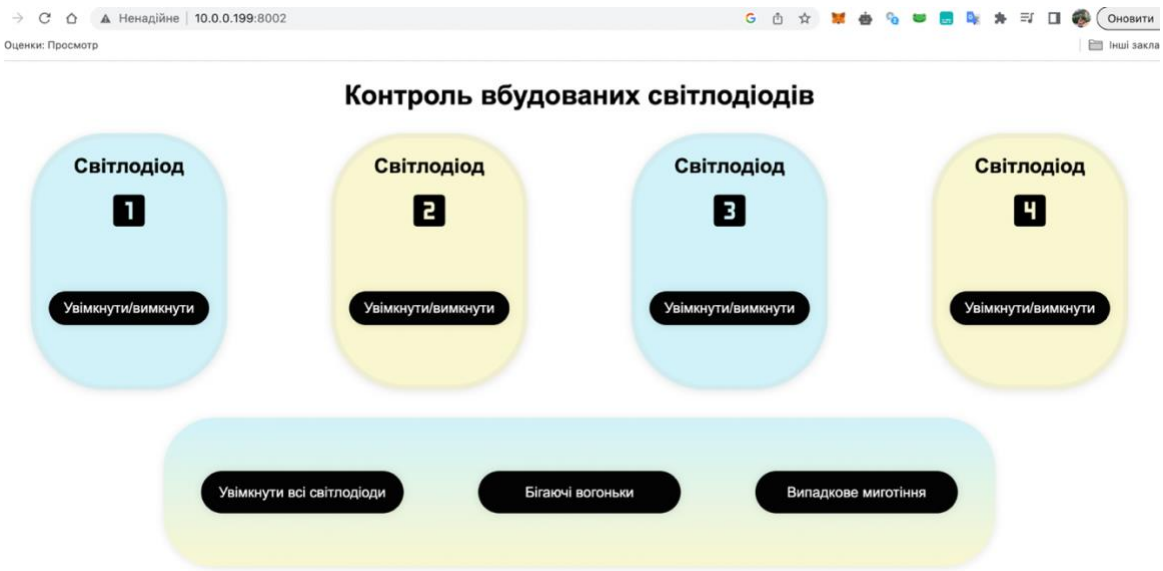


рис. 4.3.2.2 Веб-сторінка http-серверу

рис. 4.3.2.3 Вивід відповідей сервера при виконання запиту

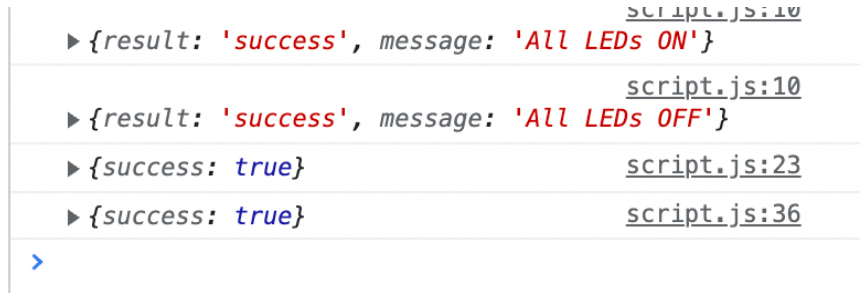


рис. 4.3.2.4 Вивід відповідей сервера при виконання запиту

4.4 Налаштування хмарного сервісу AWS для віддаленої розробки та налагодження

4.4.1 Опис процесу налаштування AWS

Створюємо обліковий запис AWS. Після входу в систему, будемо мати доступ до AWS Management Console, для налаштування різних сервісів.

Створюємо віртуальну приватну хмари (VPC), для віддаленої розробки та налагодження. VPC - це ізольована мережа в межах AWS, в якій контролюються параметри мережі.

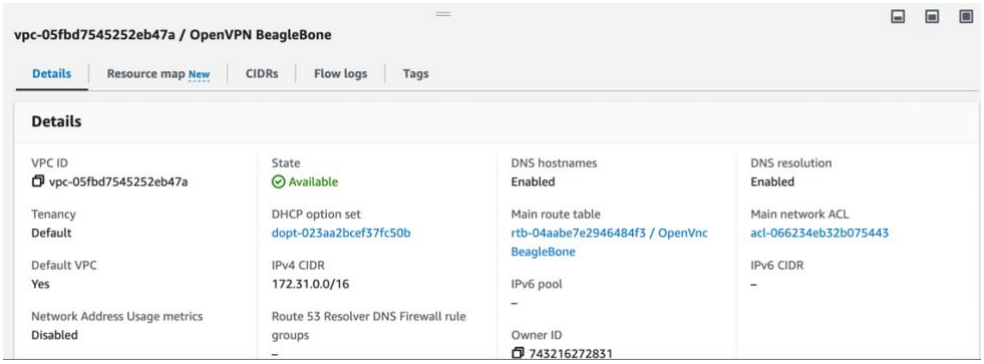


рис. 4.4.1.1 Налаштування віртуальної приватної хмари

Створюємо інстанс EC2 (Elastic Compute Cloud) - це сервіс віртуалізації, який дає можливість створення віртуальні машини в облаках AWS. Вибераємо AMI (Amazon Machine Image) - "debian-11-amd64-20230515-1381" для інстансу EC2.

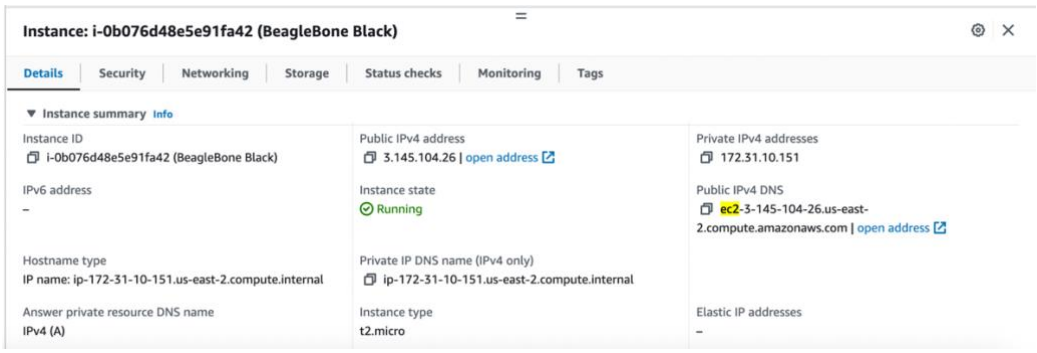


рис. 4.4.1.2 Налаштування інстанс EC2

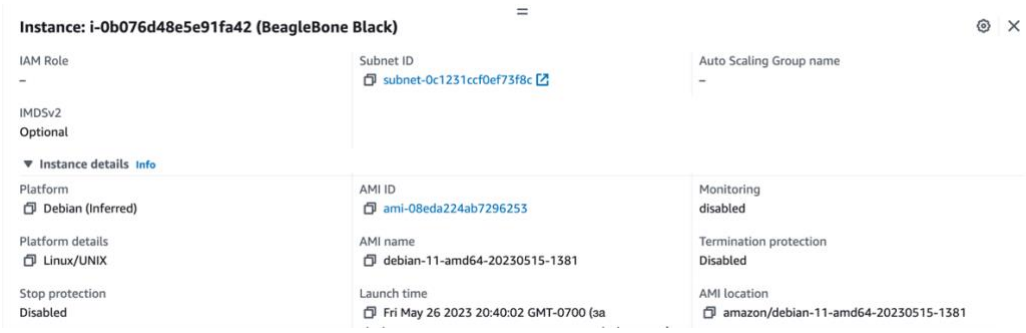


рис. 4.4.1.3 Налаштування інстанс EC2

Налаштовуємо доступ до інстансу EC2: Після створення інстансу EC2, потрібно буде налаштувати SSH ключі для безпечного доступу до інстансу.

AWS створює пару ключів, приватний ключ - my-key-pair.pem завантажується на комп'ютер, а відкритий ключ автоматично додається до інстансу EC2. Для входу до інстансу, вводимо команду “ssh -i "my-key-pair.pem" admin@ec2-3-145-104-26.us-east-2.compute.amazonaws.com”

```
debian@beaglebone:~$ ssh -i "my-key-pair.pem" admin@ec2-3-145-104-26.us-east-2.c
ompute.amazonaws.com
Linux ip-172-31-10-151 5.10.0-23-cloud-amd64 #1 SMP Debian 5.10.179-1 (2023-05-1
2) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jun  7 11:27:01 2023 from 24.87.55.22
admin@ip-172-31-10-151:~$
```

рис. 4.4.1.4 Підключення до інстансу EC2

Налаштовуємо OpenVPN, для створення безпечного каналу між локальною машиною і віртуальною машиною AWS, встановлюємо OpenVPN на клієнтську машину та на інстанс EC2. Налаштування OpenVPN включає в себе створення сертифікатів безпеки, конфігурації сервера та клієнта.

Налаштуємо сертифікати серверу, для цього створимо CA(Certificate Authority), використовуючи команду "./easyrsa build-ca", генеруємо сертифікат та ключ сервера: "./easyrsa gen-req ServerBeagleBone nopass", "./easyrsa sign-req server ServerBeagle ", генеруємо новий файл DH: "./easyrsa gen-dh"

```
admin@ip-172-31-10-151:~/easy-rsa$ ./easyrsa sign-req server ServerBeagleBone

Note: using Easy-RSA configuration from: /home/admin/easy-rsa/vars
Using SSL: openssl OpenSSL 1.1.1n  15 Mar 2022


You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 825 days:

subject=
    commonName                = ServerBeagleBone


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
```

рис. 4.4.1.5 Генерація сертифікату та ключа серверу

```
What you are about to enter is what is called a Distinguished Name or a I
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [ServerBeagleBone]:Ser
eBone

Keypair and certificate request completed. Your files are:
req: /home/admin/easy-rsa/pki/reqs/ServerBeagleBone.req
key: /home/admin/easy-rsa/pki/private/ServerBeagleBone.key


admin@ip-172-31-10-151:~/easy-rsa$ ./easyrsa gen-dh

Note: using Easy-RSA configuration from: /home/admin/easy-rsa/vars
Using SSL: openssl OpenSSL 1.1.1n  15 Mar 2022
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
.....................................................................
......+.................................+.............+.................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.............................+.......................+.................
.............................+.......................................
.....................................................................
.....................................................+...............
.....................................................................
.....................................................................
........................+....................................................
```

рис. 4.4.1.6 Генерація сертифікату та ключа серверу

Налаштовуємо конфигурацію сервера:

```
 GNU nano 5.4                  /etc/openvpn/server.conf
port 1194
proto udp
dev tun
ca /home/admin/easy-rsa/pki/ca.crt
cert /home/admin/easy-rsa/pki/issued/ServerBeagleBone.crt
key /home/admin/easy-rsa/pki/private/ServerBeagleBone.key
dh  /home/admin/easy-rsa/pki/dh.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 208.67.222.222"
push "dhcp-option DNS 208.67.220.220"
keepalive 10 120
cipher AES-256-CBC
user nobody
group nogroup
persist-key
persist-tun
status openvpn-status.log
verb 3
```

рис. 4.4.1.7 Налаштування конфігурації серверу

Тепер створимо клієнта для BeagleBone Black, та іншого комп'ютера, з якого ми будемо віддалено підключатися до плати. Створюємо запит на підпис сертифікату (CSR) для нового клієнта (див. рис. 4.4.1.8): "./easyrsa gen-req mylaptop nopass". Підпишемо CSR за допомогою нашого CA (див. рис. 4.4.1.9).

Це створить сертифікат клієнта: "./easyrsa sign-req client mylaptop". Далі треба перенести сертифікати клієнта на машину клієнта.

```
admin@ip-172-31-10-151:~$ ./easyrsa gen-req mylaptop nopass
-bash: ./easyrsa: No such file or directory
admin@ip-172-31-10-151:~$ easyrsa gen-req mylaptop nopass
-bash: easyrsa: command not found
admin@ip-172-31-10-151:~$ ls
easy-rsa
admin@ip-172-31-10-151:~$ cd ~/easy-rsa/
admin@ip-172-31-10-151:~/easy-rsa$ ./easyrsa gen-req mylaptop nopass
Using SSL: openssl OpenSSL 1.1.1n  15 Mar 2022
Generating a RSA private key
.....................................................................................+++++
.....................................+++++
writing new private key to '/home/admin/easy-rsa/pki/easy-rsa-63874.S02fNE/tmp.u8enia'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [mylaptop]:mylaptop

Keypair and certificate request completed. Your files are:
req: /home/admin/easy-rsa/pki/reqs/mylaptop.req
key: /home/admin/easy-rsa/pki/private/mylaptop.key


admin@ip-172-31-10-151:~/easy-rsa$ ./easyrsa sign-req client mylaptop
```

рис. 4.4.1.8 Створення сертифікату клієнта

```
admin@ip-172-31-10-151:~/easy-rsa$ ./easyrsa sign-req client mylaptop
Using SSL: openssl OpenSSL 1.1.1n  15 Mar 2022


You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a client certificate for 825 days:

subject=
    commonName                = mylaptop


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /home/admin/easy-rsa/pki/easy-rsa-63896.c6Q6J5/tmp.fVMC7Q
Enter pass phrase for /home/admin/easy-rsa/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'mylaptop'
Certificate is to be certified until Sep  1 04:45:16 2025 GMT (825 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /home/admin/easy-rsa/pki/issued/mylaptop.crt
```

рис. 4.4.1.9 Створення сертифікату клієнта

Налаштовуємо конфигурацию клієнту:

```
  GNU nano 3.2                                client.ovpn

client
dev tun
proto udp
remote 3.145.104.26 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca /home/debian/OpenVncClient/ca.crt
cert /home/debian/OpenVncClient/mylaptop.crt
key /home/debian/OpenVncClient/mylaptop.key
remote-cert-tls server
cipher AES-256-CBC
verb 3
```

рис. 4.4.1.10 Налаштування конфігурації клієнту

Налаштовуємо Security Groups: Security Groups у AWS виступають як віртуальний брандмауер для інстансу, щоб контролювати вхідний та вихідний трафік.[24]

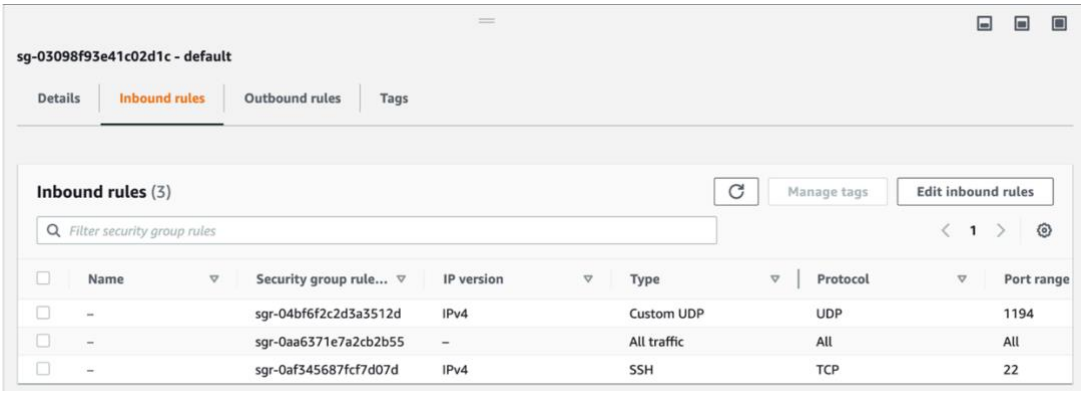


рис. 4.4.1.10 Налаштування Security Groups

4.4.2 Демонстрація роботи AWS

Запустимо сервер OpenVPN за допомогою файлу конфігурації сервера. Використовуючи команду “sudo openvpn --config /etc/openvpn/server.conf ”

```
admin@ip-172-31-10-151:~$ sudo openvpn --config /etc/openvpn/server.conf
2023-06-07 11:52:51 WARNING: --topology net30 support for server configs with IP
v4 pools will be removed in a future release. Please migrate to --topology subne
t as soon as possible.
2023-06-07 11:52:51 DEPRECATED OPTION: --cipher set to 'AES-256-CBC' but missing
 in --data-ciphers (AES-256-GCM:AES-128-GCM). Future OpenVPN version will ignore
 --cipher for cipher negotiations. Add 'AES-256-CBC' to --data-ciphers or change
 --cipher 'AES-256-CBC' to --data-ciphers-fallback 'AES-256-CBC' to silence this
 warning.
2023-06-07 11:52:51 OpenVPN 2.5.1 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4
] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on May 14 2021
2023-06-07 11:52:51 library versions: OpenSSL 1.1.1n  15 Mar 2022, LZO 2.10
2023-06-07 11:52:51 net_route_v4_best_gw query: dst 0.0.0.0
2023-06-07 11:52:51 net_route_v4_best_gw result: via 172.31.0.1 dev eth0
2023-06-07 11:52:51 Diffie-Hellman initialized with 2048 bit key
2023-06-07 11:52:51 net_route_v4_best_gw query: dst 0.0.0.0
2023-06-07 11:52:51 net_route_v4_best_gw result: via 172.31.0.1 dev eth0
2023-06-07 11:52:51 ROUTE_GATEWAY 172.31.0.1/255.255.240.0 IFACE=eth0 HWADDR=02:
cc:3e:8e:68:15
2023-06-07 11:52:51 TUN/TAP device tun0 opened
2023-06-07 11:52:51 net_iface_mtu_set: mtu 1500 for tun0
2023-06-07 11:52:51 net_iface_up: set tun0 up
2023-06-07 11:52:51 net_addr_ptp_v4_add: 10.8.0.1 peer 10.8.0.2 dev tun0
2023-06-07 11:52:51 net_route_v4_add: 10.8.0.0/24 via 10.8.0.2 dev [NULL] table
0 metric -1
2023-06-07 11:52:51 Could not determine IPv4/IPv6 protocol. Using AF_INET
2023-06-07 11:52:51 Socket Buffers: R=[212992->212992] S=[212992->212992]
2023-06-07 11:52:51 UDPv4 link local (bound): [AF_INET][undef]:1194
2023-06-07 11:52:51 UDPv4 link remote: [AF_UNSPEC]
2023-06-07 11:52:51 GID set to nogroup
2023-06-07 11:52:51 UID set to nobody
2023-06-07 11:52:51 MULTI: multi_init called, r=256 v=256
2023-06-07 11:52:51 IFCONFIG POOL IPv4: base=10.8.0.4 size=62
2023-06-07 11:52:51 ifconfig_pool_read(), in='ClientRemove3,10.8.0.4,'
2023-06-07 11:52:51 succeeded -> ifconfig_pool_set(hand=0)
2023-06-07 11:52:51 ifconfig_pool_read(), in='ClientBeaglebone,10.8.0.8,'
2023-06-07 11:52:51 succeeded -> ifconfig_pool_set(hand=1)
2023-06-07 11:52:51 IFCONFIG POOL LIST
2023-06-07 11:52:51 ClientRemove3,10.8.0.4,
2023-06-07 11:52:51 ClientBeaglebone,10.8.0.8,
2023-06-07 11:52:51 Initialization Sequence Completed
```

рис. 4.4.2.1 Запущений сервер OpenVPN

Запустимо клієнта OpenVPN на BeagleBone Black за допомогою файлу конфігурації клієнта. Використовуючи команду “sudo openvpn --config /home/debian/OpenVncClient/client.ovpn” (див. рис. 4.4.2.2)

```
debian@beaglebone:~$ sudo openvpn --config /home/debian/OpenVncClient/client.ovp
n
[sudo] password for debian:
Fri Jun  2 02:33:03 2023 OpenVPN 2.4.7 arm-unknown-linux-gnueabihf [SSL (OpenSSL
)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Apr 28 2021
Fri Jun  2 02:33:03 2023 library versions: OpenSSL 1.1.1d  10 Sep 2019, LZO 2.10
Fri Jun  2 02:33:03 2023 TCP/UDP: Preserving recently used remote address: [AF_I
NET]3.145.104.26:1194
Fri Jun  2 02:33:03 2023 Socket Buffers: R=[163840->163840] S=[163840->163840]
Fri Jun  2 02:33:03 2023 UDP link local: (not bound)
Fri Jun  2 02:33:03 2023 UDP link remote: [AF_INET]3.145.104.26:1194
Fri Jun  2 02:33:03 2023 TLS: Initial packet from [AF_INET]3.145.104.26:1194, si
d=7a149fb5 dbfdc21c
Fri Jun  2 02:33:03 2023 VERIFY OK: depth=1, CN=BeagleBoneAwsServer
Fri Jun  2 02:33:03 2023 VERIFY KU OK
Fri Jun  2 02:33:03 2023 Validating certificate extended key usage
Fri Jun  2 02:33:03 2023 ++ Certificate has EKU (str) TLS Web Server Authenticat
ion, expects TLS Web Server Authentication
Fri Jun  2 02:33:03 2023 VERIFY EKU OK
Fri Jun  2 02:33:03 2023 VERIFY OK: depth=0, CN=ServerBeagleBone
Fri Jun  2 02:33:03 2023 Control Channel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_GC
M_SHA384, 2048 bit RSA
Fri Jun  2 02:33:03 2023 [ServerBeagleBone] Peer Connection Initiated with [AF_I
NET]3.145.104.26:1194
Fri Jun  2 02:33:04 2023 SENT CONTROL [ServerBeagleBone]: 'PUSH_REQUEST' (status
=1)
Fri Jun  2 02:33:04 2023 PUSH: Received control message: 'PUSH_REPLY,redirect-ga
teway def1 bypass-dhcp,dhcp-option DNS 208.67.222.222,dhcp-option DNS 208.67.220
.220,route 10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.10 1
0.8.0.9,peer-id 0,cipher AES-256-GCM'
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: timers and/or timeouts modified
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: --ifconfig/up options modified
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: route options modified
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: --ip-win32 and/or --dhcp-option options
 modified
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: peer-id set
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: adjusting link_mtu to 1624
Fri Jun  2 02:33:04 2023 OPTIONS IMPORT: data channel crypto options modified
Fri Jun  2 02:33:04 2023 Data Channel: using negotiated cipher 'AES-256-GCM'
Fri Jun  2 02:33:04 2023 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized
 with 256 bit key
Fri Jun  2 02:33:04 2023 Incoming Data Channel: Cipher 'AES-256-GCM' initialized
 with 256 bit key
Fri Jun  2 02:33:04 2023 ROUTE_GATEWAY 10.0.0.1/255.255.255.0 IFACE=eth0 HWADDR=
98:f0:7b:60:71:61
```

рис. 4.4.2.2 Запущений клієнт OpenVPN на BeagleBone Black

Запустимо клієнта OpenVPN на іншому клієнті за допомогою файлу конфігурації клієнта. (див. рис. 4.4.2.3)

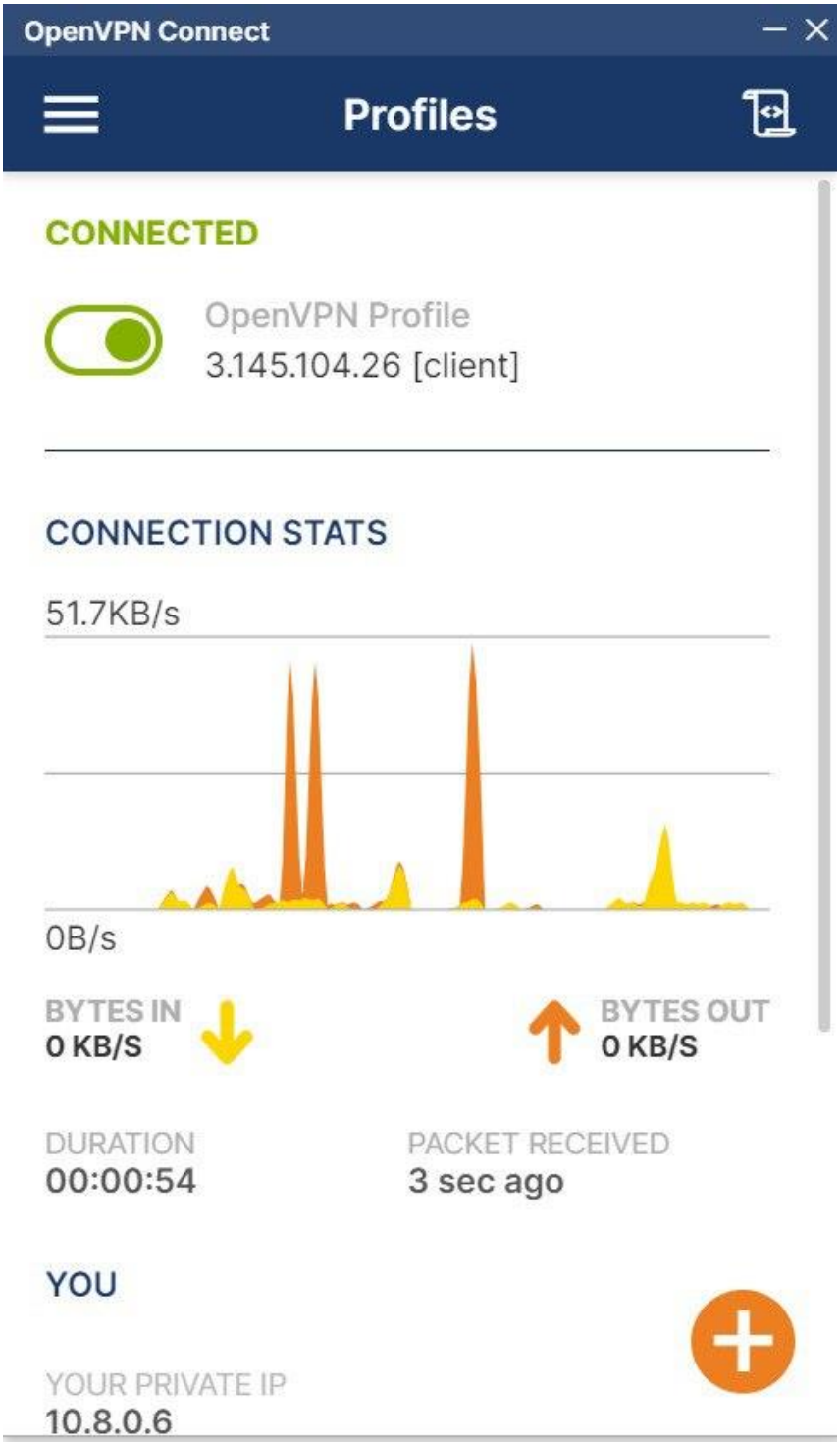


рис. 4.4.2.3 Запущений клієнт OpenVPN

Коли інший клієнт підключений до VPN, ми маємо доступ до BeagleBone Black через IP-адресу, яку він присвоїв усередині VPN. Тепер можна підключитися до плати BeagleBone Black за SSH всередині VPN.

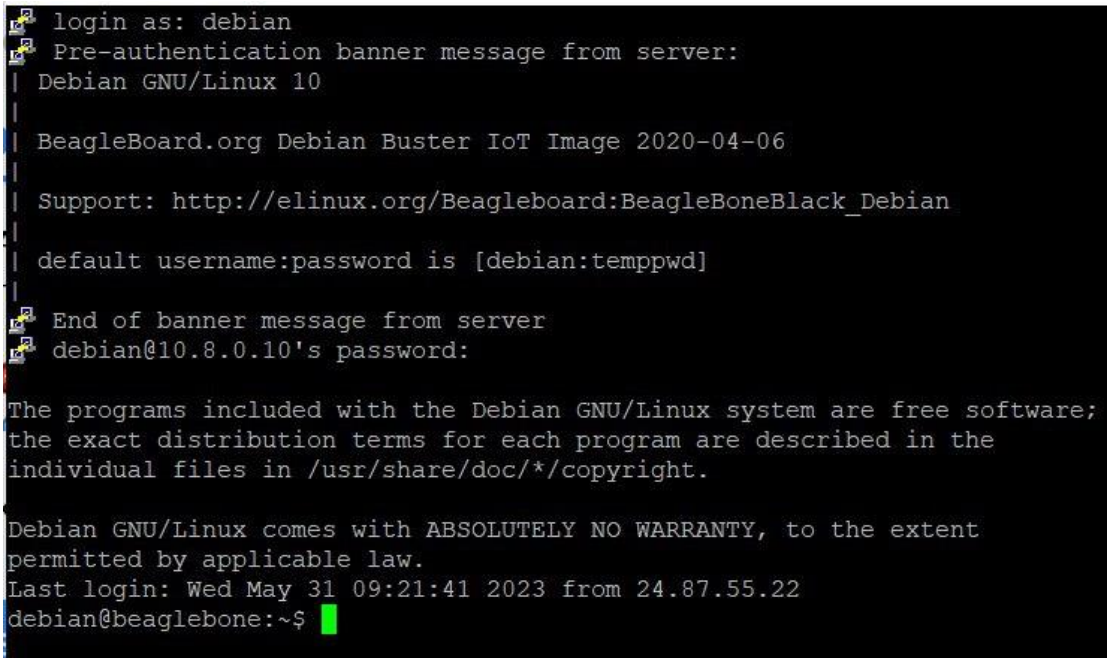


рис. 4.4.2.4 Підключитися до плати BeagleBone Black за SSH всередині VPN

Тепер ми маємо доступ з інших клієнтів, створених на сервері, до BeagleBone Black в налаштованій системі OpenVPN.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі було розглянуто процес налаштування та використання віддаленого доступу до вбудованих систем. Було проведено налаштування SSH для віддаленої роботи, описано процес налаштування SSH та показана продемонстрація його роботи.

Наступним етапом було налаштування VNC для віддаленої роботи. Відповідний процес налаштування VNC було описано, а також продемонстровано його функціональність в реальних умовах роботи.

Було виконано розробку HTTP-серверу Django для віддаленої роботи. Процес розробки HTTP-серверу було детально описано та продемонстровано його роботу.

Останнім етапом було налаштування хмарного сервісу AWS для віддаленої розробки та налагодження. Було описано весь процес налаштування AWS та продемонстровано його роботу.

Таким чином, в цьому розділі було продемонстровано ефективність використання вибраних у попередньому розділі апаратних та програмних ресурсів. Було показано, що обрані підходи до віддаленого доступу та налаштування вбудованих систем дозволяють ефективно управляти проектом, що робить обрані підходи до реалізації проекту оптимальними.

ВИСНОВКИ

Ця бакалаврська робота була присвячена дослідженню теоретичних та практичних аспектів віддаленої розробки та налагодження вбудованих систем.

У першому розділі було розглянуто основні поняття вбудованих систем, зокрема їх визначення, обмеження, а також важливість віддаленої розробки та налагодження. Особлива увага була приділена вивченню протоколів та стандартів для віддаленого доступу, засобів програмування вбудованих систем, а також питанням безпеки при віддаленому доступі до вбудованих систем.

Другий розділ присвячено огляду існуючих рішень віддаленої розробки та налагодження вбудованих систем. Він включає в себе детальний огляд методів підключення вбудованих систем до локального комп'ютера. Також було розглянуто використання SSH, VNC, JTAG, HTTP-серверу та хмарних сервісів для віддаленої роботи з вбудованими системами.

Третій розділ зосереджується на виборі та обґрунтуванні використання специфічних апаратних та програмних ресурсів для віддаленої розробки та налагодження вбудованих систем, включаючи вибір вбудованої системи, VNC, HTTP-сервера, а також хмарного сервісу.

Четвертий розділ описує налаштування та використання віддаленого доступу до вбудованих систем, зокрема налаштування SSH, VNC, HTTP-сервера, та хмарного сервісу AWS для віддаленої розробки та налагодження вбудованої системи.

Особливу увагу в даній роботі заслуговує використання хмарних технологій для віддаленої розробки та налагодження вбудованих систем. Це може забезпечити максимальну продуктивність, безпеку та зручність роботи розробників. Ця технологія виявляється особливо корисною при розробці та налаштуванні множинних вбудованих систем, як-то у випадках масового виробництва, а також у сферах де потребується одночасне керування та наголодження великої кількості вбудованих систем. Наприклад, при

налаштуванні систем у автомобільних пристроях або медичному обладнанні. Використання хмарних технологій у таких випадках може суттєво полегшити розробку, налагодження, моніторинг та управління такими вбудованими системами, покращуючи ефективність роботи та зменшуючи ризик помилок.

Завдання по повному проектуванню було виконано повністю. Розроблена система відповідає висунутим вимогам до проектування, надаючи ефективні рішення для віддаленої розробки та налагодження вбудованих систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Elecia White "Making Embedded Systems" - October 2011 - chapter 1, 2 5
2. Jonathan Valvano "Embedded Systems: Real-Time Operating Systems for Arm Cortex M Microcontrollers" - Fourth Edition, January 2017- chapter 1
3. Niclas Johansson Olof Larsson Remote Control of Embedded System Software - 2003 - chapter 3, 4
4. Bruce Powel Douglass - "Design Patterns for Embedded Systems in C: An Embedded Software Engineering Toolkit" - First edition 2011 - chapter 1
5. Tammy Noergaard - "Embedded Systems Architecture" - 2005 - chapter 1, 2 Miroslav Sveda, Radimir Vrba-INTERNET-BASED EMBEDDED SYSTEM ARCHITECTURES - 2006 - pp. 63-68
6. Derek Molloy - Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux - December 2014 - chapter 3, 10
7. Yoram Orzach, Deepanshu Khanna - Network Protocols for Security Professionals - October 2022 - chapter 2, 3, 6, 14
8. Arnold S. Berger - Debugging Embedded and Real-Time Systems - July 2020 - chapter 6, 14
9. P. Raghavan, Amol Lad, Sriram Neelakandan - Embedded Linux System Design and Development - December 2005- chapter 1, 2
10. Rui Santos, Luís Miguel Costa Perestrelo - BeagleBone For Dummies - February 2015 - chapter 1
11. Ross Anderson - Security Engineering, 3rd Edition - December 2020 - chapter 1, 27
12. David Kleidermacher, Mike Kleidermacher - Embedded Systems Security - April 2012 - chapter 1, 2, 5

13. Robert Oshana, Mark Kraeling - Software Engineering for Embedded Systems, 2nd Edition - June 2019 -chapter 1,3,14

14. Daniel J. Barrett, Richard E. Silverman, Robert G. Byrnes SSH, The Secure Shell: The Definitive Guide, 2nd Edition - May 2005- chapter 1,2

15. David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, Sailu Reddy - HTTP: The Definitive Guide - September 2002 - chapter 1,2,3

16. Technical Guide to JTAG [Електронний ресурс] // – XJTAG – Режим доступу до ресурсу: https://www.xjtag.com/about-jtag/jtag-a-technical-overview/ - 2023

17. High-level Guide to JTAG [Електронний ресурс] // – XJTAG– Режим доступу до ресурсу: https://www.xjtag.com/about-jtag/jtag-high-level-guide/ - 2023

18. What is JTAG and how can I make use of it? [Електронний ресурс] // – XJTAG – Режим доступу до ресурсу: https://www.xjtag.com/about-jtag/what-is-jtag/ - 2023

19. Django documentation [Електронний ресурс] // – Django – Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/ - 2023

20. Apache HTTP Server Documentation [Електронний ресурс] // – Apache – Режим доступу до ресурсу: https://httpd.apache.org/docs-project/- 2023

21. Nginx documentation [Електронний ресурс] // – Nginx – Режим доступу до ресурсу: https://nginx.org/en/docs/ - 2023

22. Lighttpd documentation [Електронний ресурс] // – Lighttpd – Режим доступу до ресурсу: https://redmine.lighttpd.net/projects/1/wiki/docs - 2023

23. Node.js documentation [Електронний ресурс] // – Node.js – Режим доступу до ресурсу: https://nodejs.org/en/docs - 2023

24. Amazon documentation [Електронний ресурс] // – Amazon – Режим доступу до ресурсу: https://docs.aws.amazon.com/ - 2023

25. Azure documentation [Електронний ресурс] // – Azure – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/?product=popular- 2023

26. Google Cloud documentation [Електронний ресурс] // – Google Cloud – Режим доступу до ресурсу: https://cloud.google.com/docs - 2023

27. RealVNC documentation [Електронний ресурс] // – RealVNC– Режим доступу до ресурсу: https://help.realvnc.com/hc/en-us/categories/360000301637-Documentation - 2023

28. TightVNC documentation [Електронний ресурс] // – TightVNC – Режим доступу до ресурсу: https://www.tightvnc.com/docs.php - 2023

29. UltraVNC documentation [Електронний ресурс] // – UltraVNC – Режим доступу до ресурсу: https://uvnc.com/docs/documentation-1-3-0/134-virtual-displays.html - 2023

30. Beagleboard technical guide [Електронний ресурс] // – Beagleboard.org – Режим доступу до ресурсу: https://beagleboard.org/bone - 2023

31. Mark Wilkins Learning Amazon Web Services (AWS): A Hands-On Guide to the Fundamentals of AWS Cloud - July 2019- chapter 1

32. TigerVNC documentation [Електронний ресурс] // – TigerVNC– Режим доступу до ресурсу: https://tigervnc.org/ - 2023

ДОДАТОК 1

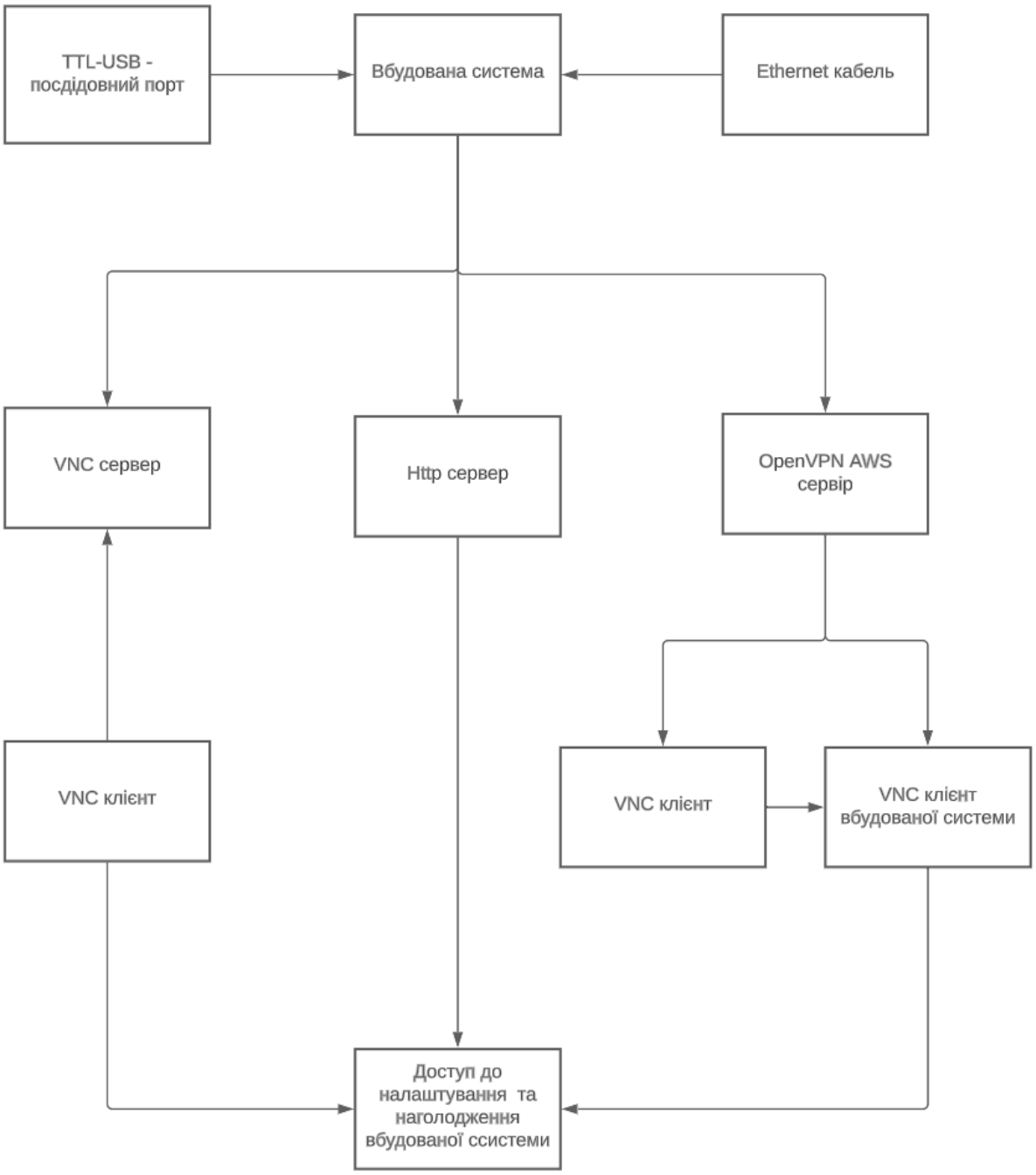
Засоби віддаленої розробки та налагодження вбудованих систем

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Слюсарь Р.О.			Алгоритм отримання віддаленого доступу до вс Структурна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ткаченко В.В.					
Н. Контр.		Виноградов Ю.М.			Дипломна работа	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Затвердив		Стіренко С.Г.					

ДОДАТОК 2

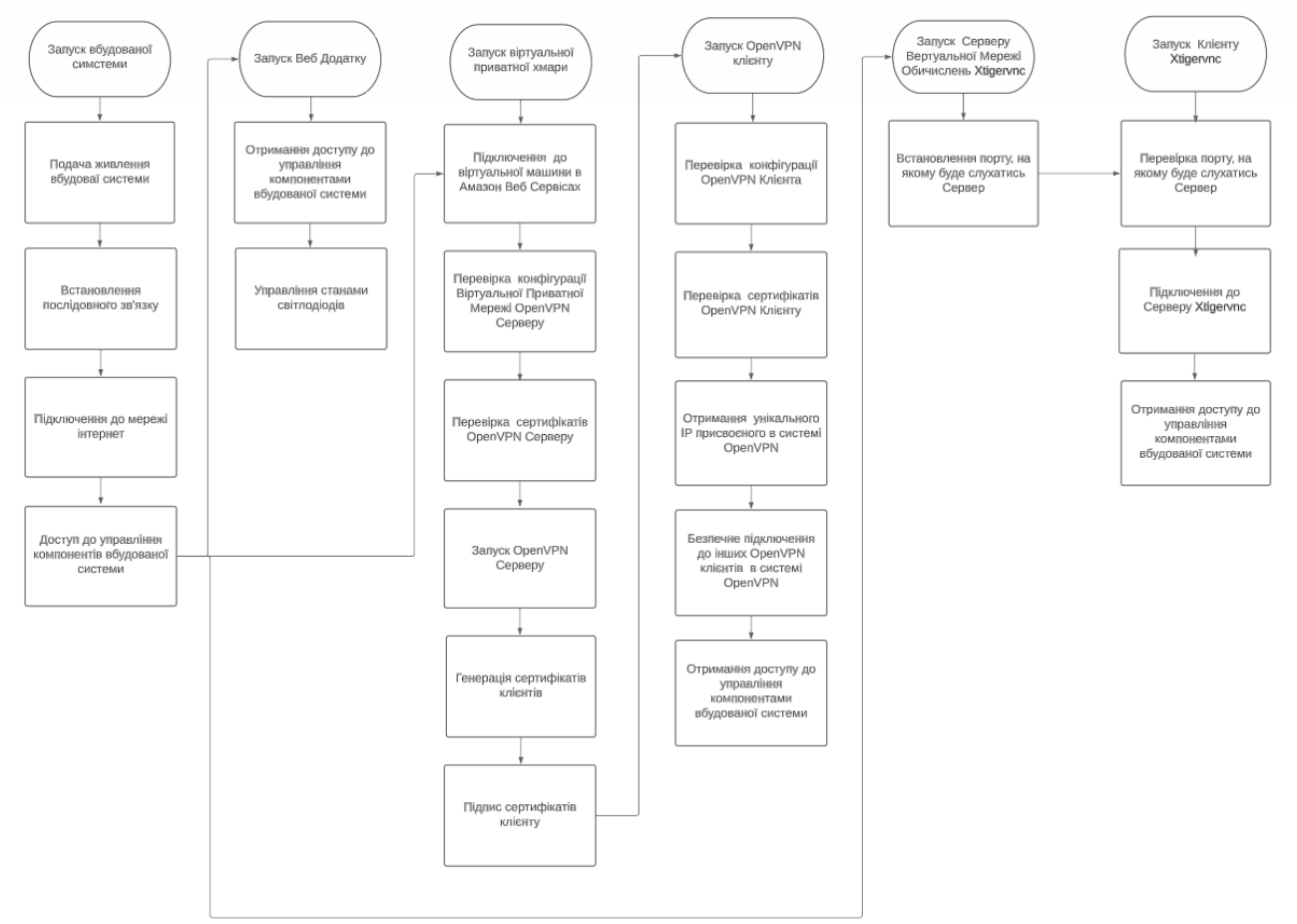
Засоби віддаленої розробки та налагодження вбудованих систем

Функціональна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Слюсарь Р.О.			Процес взаємодії між компонентами системи Функціональна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ткаченко В.В.					
Н. Контр.		Виноградов Ю.М.			Дипломна робота	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Затвердив		Стіренко С.Г.					

ДОДАТОК 3

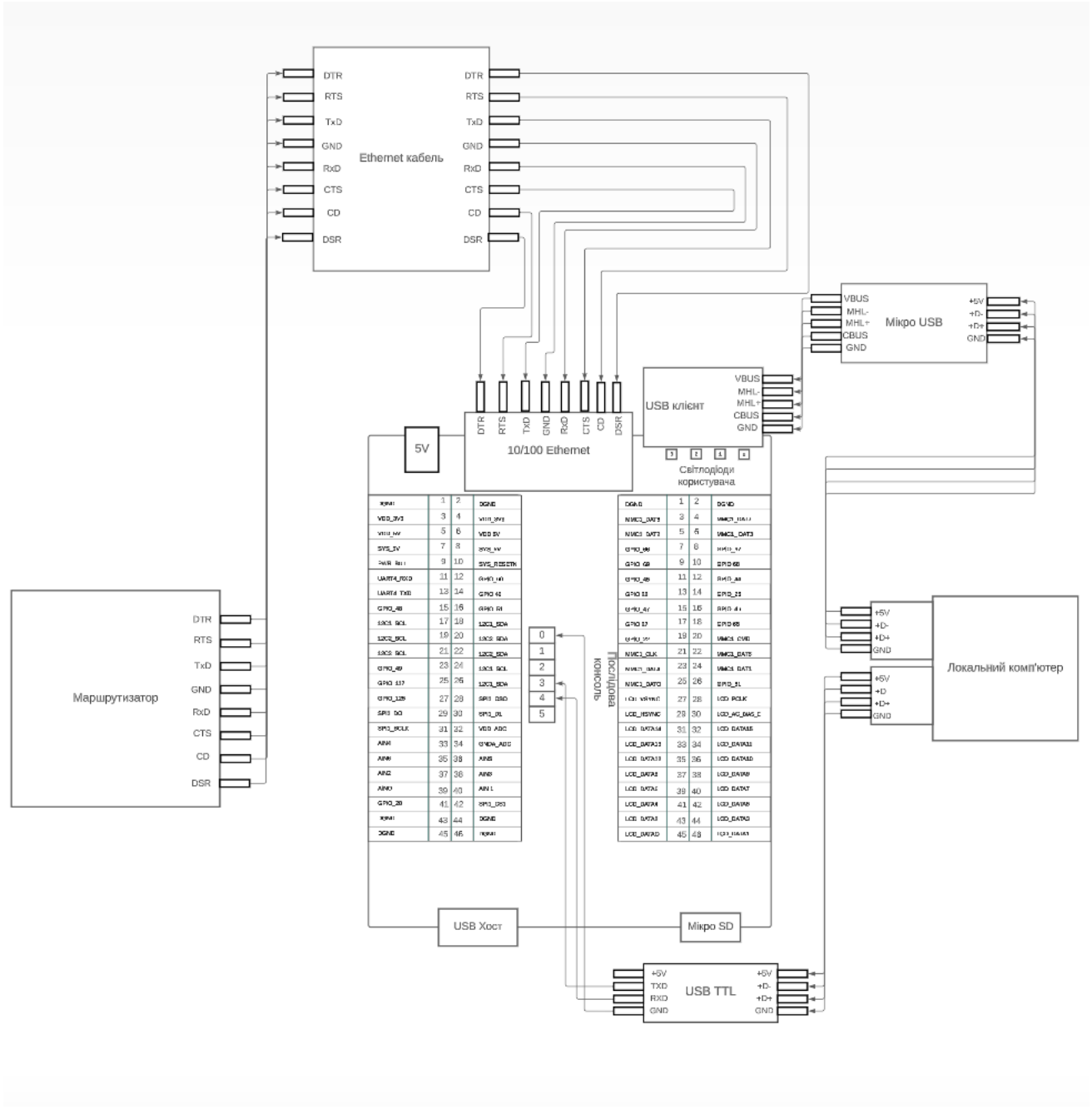
Засоби віддаленої розробки та налагодження вбудованих систем

Принципова схема

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Слюсарь Р.О.			Схема підключення вбудованої системи BeagleBone Black (Принципова схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Ткаченко В.В.				
Н. Контр.		Виноградов Ю.М.			Дипломнна робота	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93
Затвердив		Стірненко С.Г.				

ДОДАТОК 4

Засоби віддаленої розробки та налагодження вбудованих систем

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 9

Київ 2023 р

Django http server

```
from django.urls import path
from . import views

urlpatterns = [
      #URL-шаблон для кореневого шляху.
      # При зверненні до кореневого шляху буде викликатись функція index
з модуля views.
      path('', views.index, name='index'),
      #буде викликатись функція toggle_led з модуля views.
      # Значення <str:led> буде передано у функцію подання як аргумент
led.
      path('toggle/<str:led>/', views.toggle_led, name='toggle_led'),
      #Буде викликатись функція toggle_all з модуля views.
      path('toggle_all/', views.toggle_all, name='toggle_all'),
      #Буде викликатись функція running_leds з модуля views.
      path('running_leds/', views.running_leds, name='running_leds'),
      #При зверненні до цього шляху буде викликатись функція
random_leds_blink з модуля views.
      path('random_leds_blink/', views.random_leds_blink,
name='random_leds_blink'),
 ]
```

views.py

```
from django.shortcuts import render
from django.http import JsonResponse
import random
import time
import Adafruit_BBIO.GPIO as GPIO
#led_pins - це словник, який співвідносить імена
# світлодіодів (LED1, LED2, LED3, LED4) з відповідними пінами GPIO (USR0,
USR1, USR2, USR3).
led_pins = {"LED1": "USR0", "LED2": "USR1", "LED3": "USR2", "LED4":
"USR3"}

# У циклі відбувається налаштування GPIO пінів як виходи.
for pin in led_pins.values():
      GPIO.setup(pin, GPIO.OUT)

#Повертає поточний стан світлодіода
def get_led_status(led):
      return GPIO.input(led_pins[led])

#Встановлює стан світлодіода
def set_led_status(led, status):
      GPIO.output(led_pins[led], status)

#Функція обробляє запити до головної сторінки та повертає шаблон
index.html.
```

```
def index(request):
      return render(request, 'index.html')


 #Перемикає стан світлодіода

def toggle_led(request, led):

      set_led_status(led, not get_led_status(led))

      response_data = {

        "result": "success",



      "message": f"LED {led} {'ON' if get_led_status(led) else 'OFF'}",
      }
      return JsonResponse(response_data)

#Перемикає стан всіх світлодіодів
def toggle_all(request):
      current_state = GPIO.input(led_pins["LED1"])

    new_state = not current_state
      for pin in led_pins.values():
      GPIO.output(pin, new_state)

      response_data = {
      "result": "success",
      "message": f"All LEDs {'ON' if new_state else 'OFF'}",
      }
      return JsonResponse(response_data)

#Встановлює стан всіх світлодіодів
def set_all_leds(status):
      for led in led_pins:
      set_led_status(led, status)

#Вмикає кожен світлодіод послідовно з певною затримкою між ними.
def running_leds(request):
      if request.method == 'POST':
      delay = float(request.POST.get('delay', 0.1))
      for led in led_pins:
            set_led_status(led, True)
            time.sleep(delay)
            set_led_status(led, False)
      return JsonResponse({'success': True})
      else:
      return JsonResponse({'error': 'Invalid request method'})

#Вмикає і вимикає випадковий світлодіод декілька разів з певною затримкою
між кожним переключенням.
def toggle_led_without_request(led):
      set_led_status(led, not get_led_status(led))

def random_leds_blink(request):
```

```
    if request.method == 'POST':
    delay = float(request.POST.get('delay', 0.1))
    blink_count = int(request.POST.get('count', 10))

    for _ in range(blink_count):
        led = random.choice(list(led_pins.keys()))
        toggle_led_without_request(led)
        time.sleep(delay)
        toggle_led_without_request(led)

    return JsonResponse({'success': True})
    else:
    return JsonResponse({'error': 'Invalid request method'})
```

Index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Вбудована система</title>
    <link rel="stylesheet" href="{% static 'styles.css' %}">
    {% csrf_token %}
    <script>window.csrfToken = "{{ csrf_token }}";</script>
    <script src="{% static 'script.js' %}"></script>
</head>
<body>
    <h1 style="text-align: center;">Контроль вбудованих
світлодіодів</h1>
<div class="led_flex">
    <div id = "led1" class="ledClass">
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-
.9 2-2V5c0-1.1-.9-2-2-2zm-5 14h-2V9h-2V7h4v10z"></path>
    </svg>
    <button class="buttonClass"
onclick="toggleLed('LED1')">Увімкнути/вимкнути</button>
    </div>
    <div id="led2" class="ledClass">
    <p class="text">Світлодіод</p>
    <svg viewBox="0 0 24 24">
        <path d="M0 0h24v24H0z" fill="none"></path>
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-
.9 2-2V5c0-1.1-.9-2-2-2zm-4 8a2 2 0 0 1-2 2h-2v2h4v2H9v-4a2 2 0 0 1 2-
2h2V9H9V7h4a2 2 0 0 1 2 2v2z"></path>
    </svg>
    <button class="buttonClass"
onclick="toggleLed('LED2')">Увімкнути/вимкнути</button>
    </div>
```

```
      <div id = "led3" class="ledClass" >
      <p class="text">Світлодіод</p>
      <svg viewBox="0 0 24 24">
            <path d="M.01 0h24v24h-24z" fill="none"></path>
            <path d="M19.01 3h-14c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1
0 2-.9 2-2V5c0-1.1-.9-2-2-2zm-4 7.5c0 .83-.67 1.5-1.5 1.5.83 0 1.5.67 1.5
1.5V15a2 2 0 0 1-2 2h-4v-2h4v-2h-2v-2h2V9h-4V7h4a2 2 0 0 1 2
2v1.5z"></path>
      </svg>
      <button class="buttonClass"
onclick="toggleLed('LED3')">Увімкнути/вимкнути</button>
      </div>
      <div id="led4" class="ledClass" >
      <p class="text">Світлодіод</p>
      <svg viewBox="0 0 24 24">
            <path d="M0 0h24v24H0z" fill="none"></path>
            <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-
.9 2-2V5c0-1.1-.9-2-2-2zm-4 14h-2v-4H9V7h2v4h2V7h2v10z"></path>
      </svg>
      <button class="buttonClass"
onclick="toggleLed('LED4')">Увімкнути/вимкнути</button>
      </div>
</div>
<div class="allLeds" >
      <button class="allLedsButton"
onclick="toggleAll()">Увімкнути всі світлодіоди</button>
      <button class="allLedsButton" onclick="runningLeds()">Бігаючі
вогоньки</button>
      <button class="allLedsButton"




onclick="randomLedsBlink()">Випадкове миготіння</button>
</div>
</body>
</html>
```

script.js

```
//функція для перемикання стану світлодіода із зазначеним ім'ям (led).
//Вона відправляє GET-запит на шлях /toggle/<str:led>/ і виводить
отримані дані в консоль.
function toggleLed(led) {
      fetch(`/toggle/${led}/`)
      .then(response => response.json())
      .then(data => console.log(data));
}

//toggleAll() - функція перемикання стану всіх світлодіодів.
//Вона надсилає GET-запит на шлях /toggle_all/ і виводить отримані дані в
консоль
function toggleAll() {
      fetch('/toggle_all/')
      .then(response => response.json())
      .then(data => console.log(data));
```

```
}

//runningLeds() - функція для запуску вогнів, що біжать, на світлодіодах.
//Вона відправляє POST-запит на шлях /running_leds/ із затримкою (delay)
у запиті.
//Також у заголовках запиту передається CSRF-токен (X-CSRFToken),
який  використовується для захисту підробки запитів.
//Отримані дані виводяться у консоль.
function runningLeds() {
      fetch('/running_leds/', {
      method: 'POST',
      headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': window.csrfToken
      },
      body: JSON.stringify({ delay: 0.1 })
      })
      .then(response => response.json())
      .then(data => console.log(data));
}

//randomLedsBlink() - функція для випадкового миготіння світлодіодами.
//Вона відправляє POST-запит на шлях /random_leds_blink/ із затримкою
(delay) та кількістю миготінь (count) у тілі запиту
function randomLedsBlink() {
      fetch('/random_leds_blink/', {
      method: 'POST',
      headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': window.csrfToken
      },
      body: JSON.stringify({ delay: 0.1, count: 10 })
      })
      .then(response => response.json())
      .then(data => console.log(data));
}
```

styles.css

```
body {
      font-family: Arial, sans-serif;
}

#led-controls {
      display: flex;
      justify-content: space-around;
      margin-bottom: 20px;
}

#global-controls {
      display: flex;
      justify-content: space-around;
}
```

```
.led_flex {
      display: flex;
      flex-flow: row wrap-reverse;
      justify-content: space-around;
}

.ledClass {
      margin: 5px;
      padding: 5px;
      flex: 0 1 auto;
      align-self: stretch;
      border-radius: 89px;
     width: 30vh;
      height: 40vh;
}

#led1 {
      background-color: rgb(208, 242, 249);
      box-shadow: rgba(3, 3, 3, 0.1) 0px 2px 10px;
      border: 6px solid rgb(208, 236, 241);
}

#led3 {
      background-color: rgb(208, 242, 249);
      box-shadow: rgba(3, 3, 3, 0.1) 0px 2px 10px;
      border: 6px solid rgb(208, 236, 241);
}

#led2 {
      background-color: rgb(249, 247, 208);
      box-shadow: rgba(3, 3, 3, 0.1) 0px 2px 10px;
      border: 6px solid rgb(241, 238, 208);
}

#led4 {
      background-color: rgb(249, 247, 208);
      box-shadow: rgba(3, 3, 3, 0.1) 0px 2px 10px;
      border: 6px solid rgb(241, 238, 208);
}

.text {
      margin: 2vh;
      font-size: 24px;
      font-weight: 700;
      text-align: center;color: rgb(3, 3, 3);
}
svg {
      width: 50px;
      height: 50px;
      overflow: visible;
      opacity: 1;
      fill: rgb(0, 0, 0);
      display: block;
      margin: auto;
}
.buttonClass {
      width: 190px;
      height: 40px;
```

```
      background-color: rgb(3, 3, 3);
      border-radius: 25px;
      box-shadow: rgba(3, 3, 3, 0.1) 0px 0px 10px;
      border: 0px;
      font-size: 17px;
      text-align: center;
      color: white;
      margin: 0 auto;
      display: block;
      margin-top: 10vh;
}

.allLeds{
      display: flex;
      justify-content: space-around;
      align-items: center;
      width: 140vh;
      height: 25vh;
      margin: auto;
      margin-top: 4vh;
      background: linear-gradient(#D0F2F9, #F9F7D0);
      border-radius: 61px;
      box-shadow: rgba(3, 3, 3, 0.1) 0px 2px 10px; border: 0px;

}

.allLedsButton {
      display: flex;
      align-items: center;
      justify-content: center;
      box-sizing: border-box;
      overflow: hidden;
      outline: none;
      cursor: inherit;
      width: 240px;
      height: 50px;
      opacity: 1;
      background-color: rgb(3, 3, 3);
      border-radius: 25px;
      box-shadow: rgba(3, 3, 3, 0.1) 0px 0px 10px;
      border: 0px;
      text-align: center;
      color: white;
      font-size: 17px;

}
```

OpenVPN

Server.conf

```
//Визначає порт, на якому OpenVPN сервер слухає вхідні з'єднання.
port 1194
```

```
//Визначаємо, що протокол UDP, використовується OpenVPN.
proto udp
// Використовуємо TUN інтерфейс, який дозволяє маршрутизувати IP
//трафік.
dev tun
// Шлях до файлу сертифіката центру сертифікації.
ca /home/admin/easy-rsa/pki/ca.crt
//Шляхи до сертифіката сервера та відповідного приватного ключа.
cert /home/admin/easy-rsa/pki/issued/ServerBeagleBone.crt
 key /home/admin/easy-rsa/pki/private/ServerBeagleBone.key
//Шлях до файлу з параметрами Diffie-Hellman, які використовуються для
//обміну ключами.
dh /home/admin/easy-rsa/pki/dh.pem
//Визначається  підсітка IP-адрес, які надаються клієнтам VPN.
server 10.8.0.0 255.255.255.0
// Налаштування зберигання  прив'язку між IP-адресою клієнта та його
//Common Name в файлі ipp.txt.
ifconfig-pool-persist ipp.txt
//Керування всим трафіком клієнта через VPN.
push "redirect-gateway def1 bypass-dhcp"
//Встановлення DNS-сервери для клієнтів VPN.
push "dhcp-option DNS 208.67.222.222"
 push "dhcp-option DNS 208.67.220.220"
keepalive 10 120
// Алгоритм шифрування, який використовується для захисту даних.
cipher AES-256-CBC
// Запуск серверу VPN від імені користувача "nobody" і групи "nogroup"
//для покращення безпеки.
user nobody
group nogroup
//Зберігаємо  ключ та TUN/TAP інтерфейс при перепідключенні.
persist-key
 persist-tun
//Вказується шлях до файлу, в якому зберігається статус сервера
status openvpn-status.log
//Регулюється рівень деталізації журналів
verb 3

client.ovnp
client
// Використовуємо "tun" (tunnel) режим, який дозволяє передачу IP пакетів
через VPN.
dev tun
// Використовуємо протокол UDP для з'єднання з сервером.
proto udp
// Вказуємо IP-адресу та порт сервера OpenVPN.
remote 3.145.104.26 1194
// Перез'єднання з сервером, якщо з'єднання перерветься. Процес
повторюється нескінченно.
resolv-retry infinite
// Ця опція забороняє зв'язувати деякі локальні адреси та порти.
nobind
// Забезпечує, що ключі залишаються незмінними після перепідключення.
persist-key
// Забезпечує, що TUN/TAP інтерфейс залишається доступним після
перепідключення.
persist-tun
// Запитуємо сертифікат віддаленого сервера для автентифікації.
```

```
remote-cert-tls server
// Визначаємо алгоритм шифрування (AES-256-CBC) для захисту даних.
cipher AES-256-CBC
// Встановлюємо рівень деталізації у журналі.
verb 3
//Сертифікат авторитету сертифікації (CA), використовується для перевірки
сертифіката сервера.
<ca>
-----BEGIN CERTIFICATE-----
//видалені задля безпеки
-----END CERTIFICATE-----
</ca>
// Вміст сертифіката клієнта
<cert>
-----BEGIN CERTIFICATE-----
//видалені задля безпеки
</cert>
// Вміст приватного ключа клієнта
<key>
-----BEGIN PRIVATE KEY-----
//видалені задля безпеки
-----END PRIVATE KEY-----
</key>
```