КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра медичної радіофізики

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри медичної радіофізики
від____.____.2024 року, протокол №_____
Завідувач кафедри канд. фіз.-мат. наук, доцент
_____Сергій РАДЧЕНКО

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
на тему:

«ОБРОБКА ЕЛЕКТРОЕНЦЕФАЛОГРАФІЧНИХ СИГНАЛІВ З ВИКОРИСТАННЯМ ПЛІС XILINX SPARTAN»

Виконав:
студент 4-го курсу
денної форми навчання
спеціальності 105 Прикладна фізика та наноматеріали
ОПП «Електроніка та інформаційні технології в медицині»
Рижевський Євген Русланович_____

Науковий керівник:
канд. фіз.-мат. наук, доцент
Судаков Олександр Олександрович_____

Рецензент:
канд. фіз.-мат. наук, доцент
Бойко Юрій Володимирович_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент_____ Євген РИЖЕВСЬКИЙ

Київ – 2024

СПИСОК СКОРОЧЕНЬ

ПЛІС – програмована логічна інтегральна схема

FPGA – Field-Programmable Gate Array (поле-програмована матриця логічних елементів)

HDL – Hardware Description Language (мова опису апаратних засобів)

IDE – Integrated Development Environment (інтегроване середовище розробки)

XDC – Xilinx Design Constraints або Xilinx Design Constraints File (файл обмежень дизайну Xilinx)

FFT IP-core – Fast Fourier Transform Intellectual Property core (ядро інтелектуальної власності швидкого перетворення Фур'є)

FTDI – Future Technology Devices International (шотландська приватна компанія з виробництва напівпровідникових пристроїв, що спеціалізується на технології універсальної послідовної шини (USB))

UART – Universal Asynchronous Receiver-Transmitter (універсальний асинхронний приймач/передавач)

FIFO – First in, first out (першим прийшов - першим пішов)

JTAG – Joint Test Action Group (робоча група з розробки стандарту IEEE 1149)

SPI – Serial Peripheral Interface (послідовний периферійний інтерфейс)

EEPROM – Electrically Erasable Programmable Read-Only Memory (постійний запам'ятовувач, що програмується та очищується за допомогою електрики, один з видів енергонезалежної пам'ят)

LUT – Look-Up-Table

CLB – Configurable logic blocks (конфігуровані логічні блоки)

DSP – Digital signal processor (цифровий процесор обробки сигналів)

LVCMOS – Low voltage complementary metal oxide semiconductor (низьковольтний комплементарний металооксидний напівпровідник)

LVDS – Low-voltage differential signaling (низьковольтна диференціальна передача сигналів)

USB – Universal Serial Bus (універсальна послідовна шина)

МРТ – магнітно-резонансна томографія

КТ – Комп’ютерна томографія

РЕФЕРАТ

Кваліфікаційна робота бакалавра 53 стор., 17 рисунків, 6 таблиць, 12 джерел

Реалізовано алгоритм та прототип приладу для аналізу електроенцефалографічних (ЕЕГ) даних за допомогою ПЛІС Xilinx Spartan-7 з метою визначення відкритих та закритих очей людини за цими сигналами. Виконано тестування пристрою. Надано опис етапів розробки та результатів роботи пристрою.

Ключові слова: СИСТЕМА АНАЛІЗУ ДАНИХ, ПЛІС, FPGA(FIELD-PROGRAMMABLE GATE ARRAY), XILINX SPARTAN-7, CMOD-S7, VIVADO DESIGN SUITE, AMD XILINX, VERILOG, HDL, БІТСРІМ-ФАЙЛ, ФАЙЛ XDC(XILINX DESIGN CONSTRAINTS), PYTHON-СКРИПТ, FTDI(FUTURE TECHNOLOGY DEVICES INTERNATIONAL), FFT(FAST FOURIER TRANSFORM) IP-ЯДРО, UART(UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER).

ЗМІСТ

ВСТУП

В епоху стрімкого розвитку цифрових технологій, одним із ключових завдань є розробка і застосування програмованих логічних інтегральних схем (ПЛІС). Програмовані логічні інтегральні схеми дають можливість програмувати внутрішню схему цифрових і цифро-аналогових пристроїв. Реалізація дуже швидких обчислювальних систем можлива лише за допомогою ПЛІС завдяки гнучкості і паралелізму. Цифрові системи з фіксованою внутрішньою структурою (процесори, мікроконтролери) не дають можливості досягти швидкостей обчислень і простоти схем, які можна отримати за допомогою ПЛІС. Також актуальною задачею є створення та оптимізації різних алгоритмів аналізу даних на основі ПЛІС.

Застосування ПЛІС надзвичайно актуальне для створення мобільних медичних пристроїв, таких як засоби детектування та прогнозування епілептичних нападів, серцевих нападів, тощо.

Дана робота присвячена реалізації простої схеми аналізу та обробки електроенцефалографічних даних на базі ПЛІС Xilinx Spartan-7, одного з передових пристроїв у цій галузі. Дана система, повинна робити визначення відкритих та закритих очей за сигналами ЕЕГ використовуючи дану модель ПЛІС. Виконано тестування можливостей цього пристрою з метою подальшої побудови даної системи аналізу медичних даних. ПЛІС Xilinx Spartan-7 Cmod S7 являє собою високопродуктивне та економічне рішення для цифрового оброблення даних. Мова програмування Verilog, використана у цій роботі, є стандартом галузі для опису апаратних засобів, що робить її застосування вкрай актуальним.

1. Програмовані логічні інтегральні схеми

1.1. Xilinx Spartan-7 Cmod S7

Програмована логічна інтегральна схема (ПЛІС) Xilinx Spartan-7 Cmod S7 являє собою високопродуктивне, економічне рішення для цифрового оброблення даних. Ця серія ПЛІС від Xilinx характеризується високою щільністю логічних блоків, енергоефективністю та можливістю роботи з різними типами вхідних і вихідних сигналів. Spartan-7 особливо примітна своєю здатністю до швидкої реалізації цифрових схем, що робить її ідеальною для освітніх проєктів і невеликих розробок.

Архітектура та особливості Xilinx Spartan-7 Cmod S7:

- Структура LUT: архітектура Spartan-7 FPGA заснована на LUT. LUT є основними компонентами, які використовуються для реалізації логіки в FPGA. У Spartan-7 Cmod S7 дані LUT організовані як ієрархічна структура, що забезпечує ефективну реалізацію логіки.
- CLB (конфігуровані логічні блоки): ці блоки містять суміш LUT, тригерів і логіки перенесення, які разом забезпечують конфігуровані ресурси для реалізації різних функцій.
- Фрагменти DSP: сегменти цифрової обробки сигналів (DSP) – це спеціальні блоки, оптимізовані для роботи математичних функцій, які зазвичай використовуються в обробці сигналів. Spartan-7 Cmod S7 FPGA має кілька байтів DSP, які можна використовувати для таких цілей, як фільтрація, згортка та кореляція.
- Можливості вводу/виводу: Spartan-7 Cmod S7 FPGA має кілька варіантів введення/виведення, включаючи такі стандарти, як LVCMOS, LVDS і USB. Ці універсальні можливості введення/виведення забезпечують взаємодію з різними зовнішніми пристроями та датчиками, які зазвичай використовуються в медицині.

Застосування Xilinx Spartan-7 Cmod S7:

- Аналіз та обробка даних ЕЕГ: ПЛІС Xilinx Spartan-7 Cmod S7 можна використовувати в системах обробки даних ЕЕГ (електроенцефалограма) для виконання завдань у режимі реального часу, таких як обробка сигналів, виділення ознак і розпізнавання образів.
- Медичні системи візуалізації: ПЛІС можуть підвищити ефективність медичних систем візуалізації, таких як сканери МРТ і КТ, що підвищить якість і швидкість зображень.
- Портативні пристрої моніторингу: низьке енергоспоживання та висока швидкість обробки ПЛІС Xilinx Spartan-7 Cmod S7 є перевагою для портативних медичних пристроїв моніторингу та аналізу життєво важливих показників в реальному часі.

Архітектура Spartan-7 Cmod S7 FPGA, а також її особливості є корисними для розробників, які хочуть створювати інноваційні рішення проблем у сфері охорони здоров'я[3].

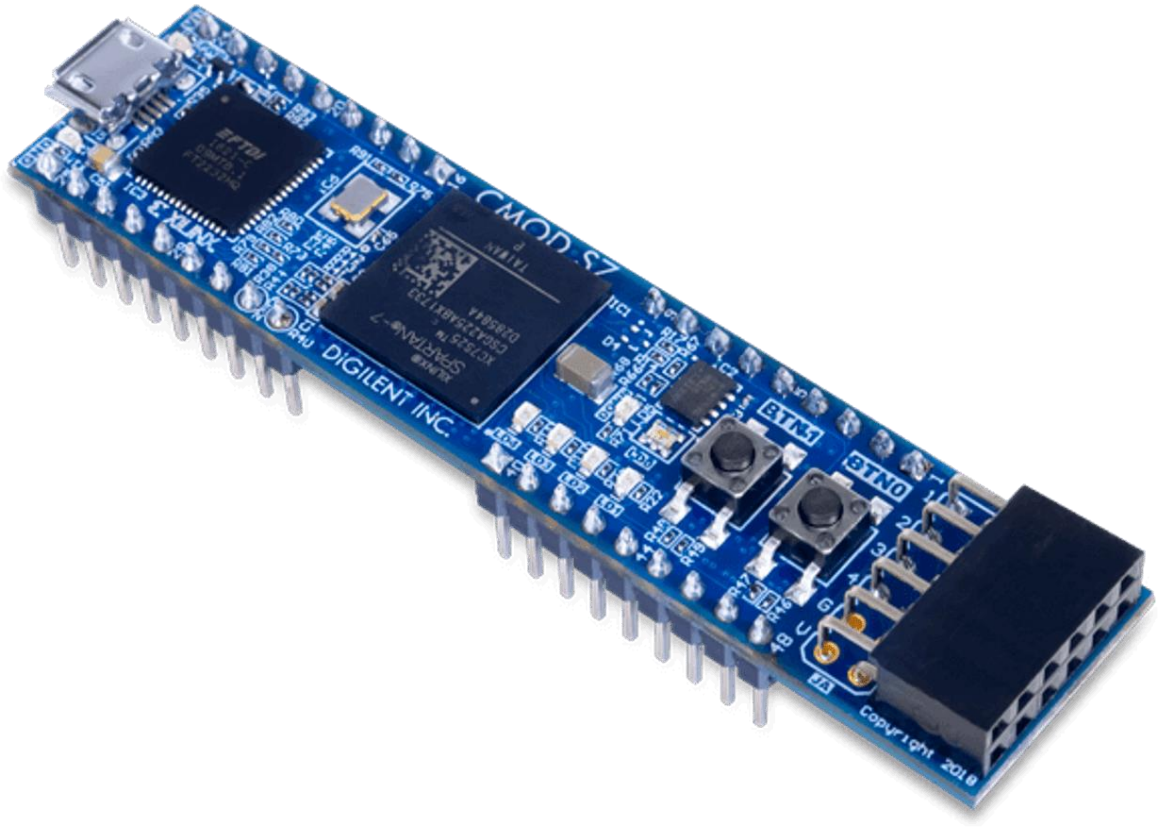

Рис.1.1 Фото Xilinx Spartan-7 Cmod S7(вид зверху)

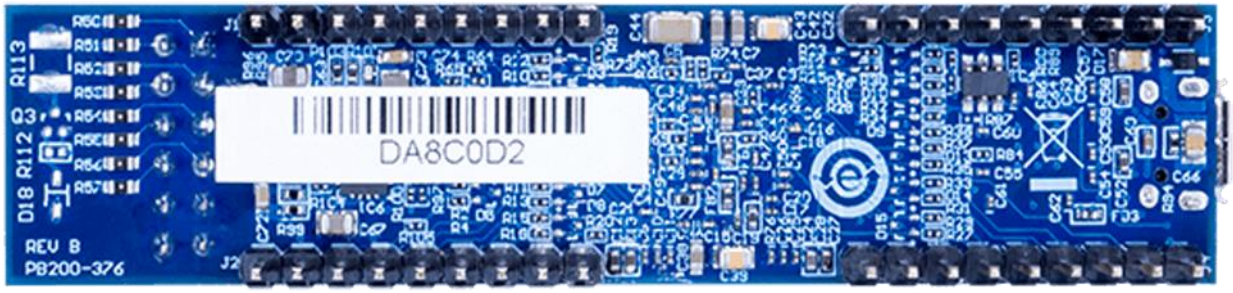

Рис.1.2 Фото Xilinx Spartan-7 Cmod S7(вид знизу)

1.2. FTDI232

FTDI232 — це сімейство універсальних мікросхем, створених Future Technology Devices International (FTDI), які полегшують перетворення USB на інші інтерфейси, такі як UART, FIFO, JTAG і SPI. Серія мікросхем FTDI232R користується особливою популярністю завдяки простому дизайну та великій кількості функцій.

Основні атрибути FTDI232:

- Підтримка повної швидкості USB 2.0: максимальна швидкість передачі даних становить до 12 Мбіт/с.
- Інтегрований UART: асинхронний послідовний зв'язок з апаратною підтримкою для керування потоком даних (RTS/CTS).
- Виявлення швидкості передачі даних: автоматичне розпізнавання швидкості передачі даних від 300 біт/с до 3 Мбіт/с.
- Підтримка різного кодування даних: 7 або 8-бітні дані, 1 або 2-бітна зупинка, парність.
- Програмований EEPROM: здатність змінювати параметри за допомогою внутрішнього EEPROM, що дає змогу налаштувати мікросхему для певної мети.
- Живлення від USB: не вимагає окремого джерела живлення, що зменшує складність інтеграції в пристрій.

Застосування FTDI232:

- Створення невеликих, недорогих комп’ютерів, які використовують мікроконтролери для зв’язку з комп’ютером через USB.

- Створення та тестування цифрових систем.
- Використовуйте в аматорських проектах, які хочуть створити послідовне з'єднання через USB[6].

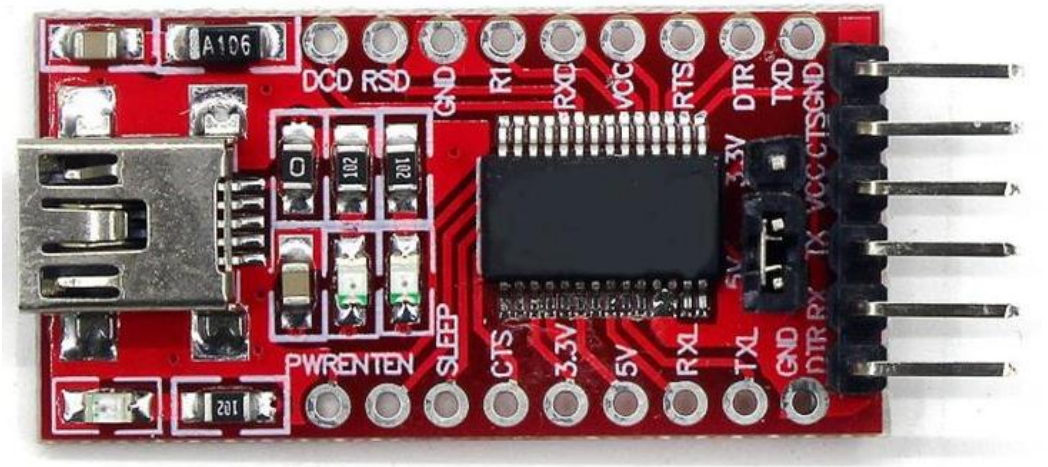

Рис.1.3 Фото FTDI232(вид зверху)

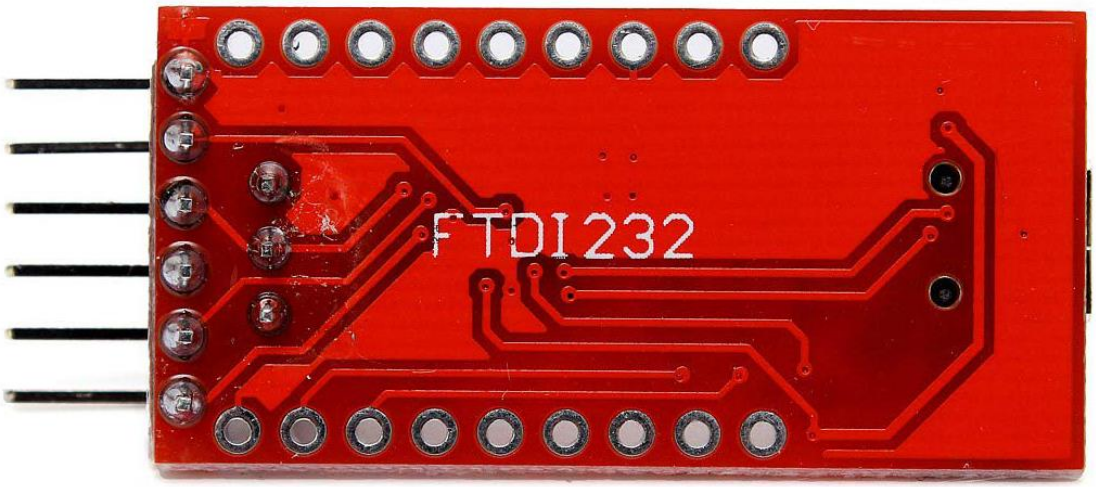

Рис.1.4 Фото FTDI232(вид знизу)

1.3. Інтерфейс UART

UART (Universal Asynchronous Receiver-Transmitter) - це апаратний пристрій або модуль, який здійснює асинхронний послідовний обмін даними між комп'ютером та периферійним пристроєм. UART- один з найпоширеніших методів послідовного зв'язку в цифрових системах.

Основні характеристики UART:

- Асинхронна передача даних: не вимагає синхронізації тактового сигналу між передавачем і приймачем, що спрощує підключення.

- Підтримка різних швидкостей передачі: дозволяє встановити швидкість обміну даними від декількох сотень біт/ сек до декількох мегабіт / с.
- Формат даних: підтримка 5-9 біт даних, 1 або 2 стоп-біта, парність (парна,непарна або відсутня).
- Управління потоком даних: апаратне (RTS/CTS) або програмне (XON/XOFF) управління потоком для запобігання втрати даних.
- Простота реалізації: широко підтримується багатьма мікроконтролерами, мікропроцесорами та іншими цифровими пристроями.

Застосування:

- Послідовний зв'язок між комп'ютером і периферійними деками.
- Підключення датчиків та інших модулів до мікроконтролерів.
- Налагодження та діагностика цифрових систем[7].

1.4. FFT IP-ядро

FFT IP-ядро (Fast Fourier Transform Intellectual Property Core) - це спеціалізований апаратний модуль, призначений для виконання швидких перетворень Фур’є. Це ядро, як правило, включено в FPGA (Field Programmable Gate Array) для сприяння швидшій обробці сигналу.

Основні атрибути ядра FFT IP:

- Висока продуктивність: на апаратному рівні реалізація перетворення Фур'є швидка, це значно перевищує швидкість програмних реалізацій.
- Настроювана точність: підтримка фіксованих або плаваючих обчислень, це допоможе оптимізувати використання ресурсів.
- Універсальність конфігурації: можливість змінювати розмір ШПФ (наприклад, 128, 256, 1024 точки) відповідно до потреб конкретної програми.
- Мінімізація затримки: ідеально підходить для використання в режимі реального часу через мінімальну затримку обробки сигналу.
- Інша системна інтеграція: підтримка загальних інтерфейсів, які можна використовувати для інтеграції інших компонентів і систем.

Застосування:

- Аналіз спектру сигналу в реальному часі.
- Обробка зображень і відео.
- Системи радіозв'язку і радари.
- Медичні зображення(наприклад, магнітно-резонансна томографія)[8].

1.5. Мова Verilog

Verilog - це мова опису апаратури (HDL), яку широко використовують у проєктуванні та реалізації цифрових систем на ПЛІС. Ця мова дає змогу розробникам ефективно описувати структуру та поведінку електронних систем на високому рівні абстракції. Verilog здатна моделювати як дуже прості, так і надзвичайно складні цифрові системи, роблячи її затребуваною в промисловості та академічних колах.

Синтаксис Verilog та її конструкції:

- Модулі: основними блоками Verilog є модулі, які інкапсулюють певні функціональні можливості цифрової системи. Модулі можуть представляти що завгодно, від простих логічних елементів до складних підсистем.
- Порти: модулі можуть мати вхідні, вихідні або двонаправлені порти, які визначають, як дані надходять до модуля та виходять з нього. Порти оголошуються за допомогою ключових слів input, output або inout.
- Типи даних: Verilog підтримує різні типи даних, включаючи провідні, регулярні, цілі, дійсні та параметри. Ці типи даних використовуються для представлення сигналів, змінних і констант у проекті.
- Керуючі структури: Verilog включає, також, керуючі структури, такі як оператори if, else, case, for і while, що дозволяє умовне виконання та ітерації циклу в описі обладнання[4].

1.6. Огляд програмного середовища Vivado

Vivado Design Suite від Xilinx - це інтегроване середовище розробки (IDE), призначене для створення й аналізу HDL-проектів, особливо для ПЛІС сімейства Xilinx. Vivado пропонує вдосконалені можливості проєктування, включно з графічним інтерфейсом, засобами аналізу та налагодження, а також поліпшену підтримку для синтезу й оптимізації коду. Це середовище розробки дає змогу значно прискорити процес проектування і спростити тестування проектів на ПЛІС, що робить його незамінним інструментом для сучасних розробників.

Робочий процес у Vivado має декілька етапів:

- Створення проекту: процес проектування починається зі створення нового проекту у Vivado. Вибір цільового пристрою ПЛІС, установка параметрів проекту та впорядкування файлів дизайну.
- Введення дизайну: введення проектів за допомогою одного з декількох методів, наприклад, написання коду HDL, використання графічних інструментів, таких як IP Integrator, або імпортування існуючих ядер IP.
- Симуляція: моделювання проекту, задля перевірки його функціональності та продуктивності. Vivado містить вбудовані інструменти моделювання, такі як Vivado Simulator, який дозволяє моделювати як поведінку, так і час.
- Синтез: синтез проекту, в якому код HDL перетворюється на список з’єднань логічних вентилів і тригерів. Vivado оптимізує дизайн для площі, потужності та продуктивності на даному етапі, синтезу.
- Реалізація: під час впровадження синтезований дизайн відображається на цільовому пристрої, що передбачає розміщення та маршрутизацію логічних елементів на фізичних ресурсах ПЛІС. Vivado містить розширені алгоритми для оптимізації часу, потужності та використання під час цього процесу.
- Перевірка: після впровадження виконується додаткова перевірка, аби переконатися, що проект відповідає вимогам щодо часу та працює належним чином.

- Генерація бітового потоку: генерація бітового файлу, що містить конфігураційні дані для ПЛІС. Цей файл можна використовувати для програмування ПЛІС та налаштування його для реалізації бажаної функції[2].

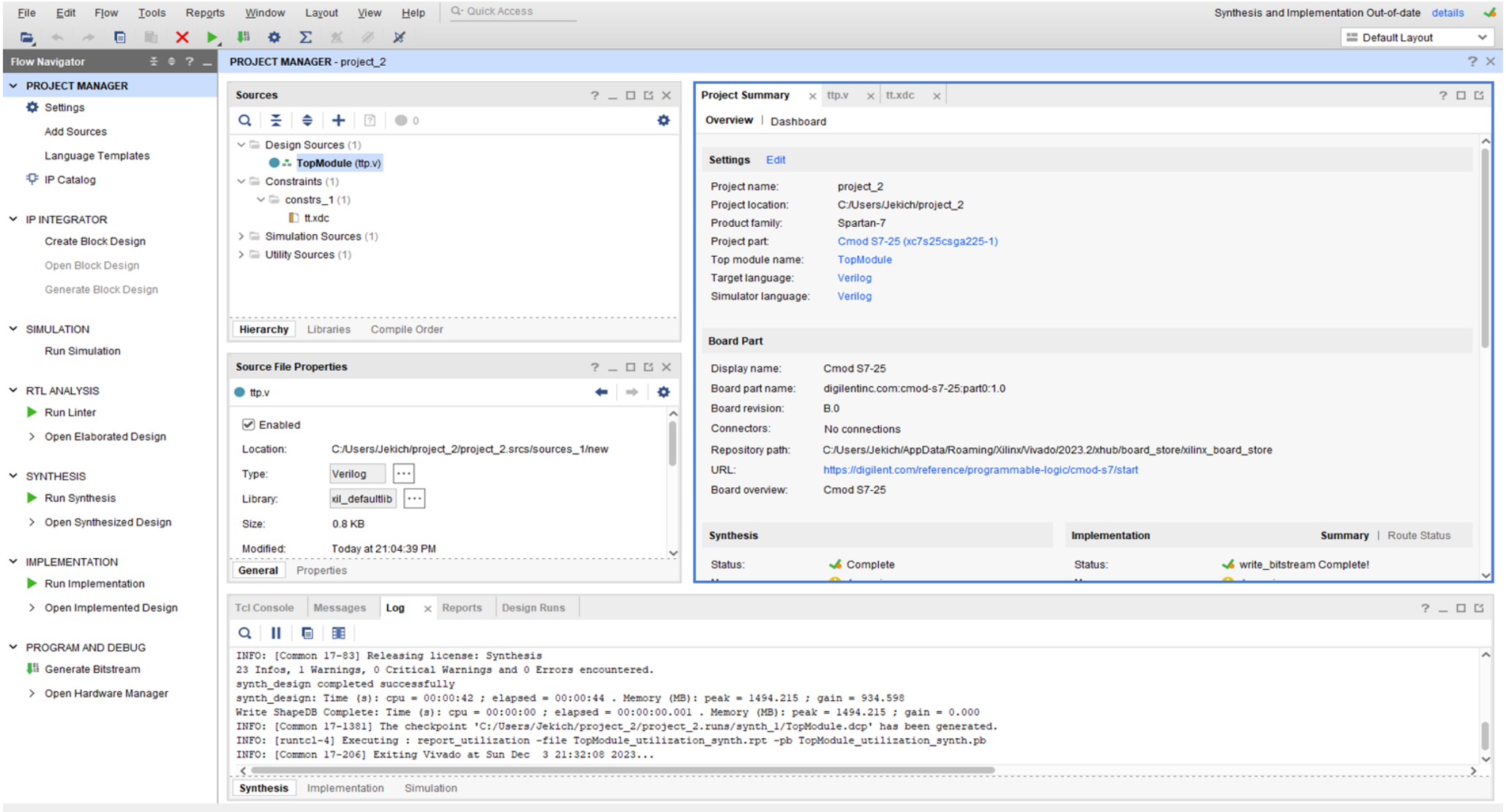

Рис.1.5 Інтерфейс Vivado

1.7. Постановка задачі

Вивчення мови Verilog і роботи з ПЛІС є актуальним з кількох причин:

- Версатильність**:** Verilog дає змогу ефективно описувати апаратні засоби, що робить його невід'ємною частиною проектування сучасних електронних пристроїв.
- Інноваційність**:** Xilinx Spartan-7 являє собою сучасну ПЛІС, що має високу продуктивність і гнучкість, що робить її ідеальною для освітніх і дослідницьких цілей.
- Практична значущість: навички роботи з Verilog і ПЛІС важливі для інженерів-електронників, оскільки їх широко застосовують у промисловості та дослідженнях.

Цей проєкт є важливим кроком у навчанні та розвитку навичок, необхідних для успішної кар'єри в галузі електронної інженерії та цифрової обробки даних.

Основною метою цього проєкту було розроблення та тестування системи аналізу електроенцефалографічних даних для визначення відкритих та закритих очей за сигналами ЕЕГ використовуючи ПЛІС Xilinx Spartan-7.

Для досягнення цієї мети було поставлено такі завдання:

- Вивчення й освоєння програмного середовища Vivado для роботи з ПЛІС Xilinx Spartan-7.
- Розробка вихідного коду мовою Verilog для реалізації заданої функціональності.
- Створення файлу XDC для налаштування параметрів ПЛІС.
- Генерація і його завантаження на ПЛІС.
- Створення Python-скрипту для зчитування ЕЕГ даних з наданого файлу та відправки цих даних через інтерфейс UART, для подальшої обробки на ПЛІС Xilinx Spartan-7.
- Аналіз результатів і оцінка ефективності розробленої системи.
- Актуальність вивчення мови Verilog і роботи з ПЛІС Xilinx Spartan-7.

2. ПРОГРАМУВАННЯ ПЛІС

2.1. Детальний опис процесу завантаження та встановлення Vivado

Для реалізації проекту на ПЛІС Xilinx Spartan-7 була обрана програмна платформа Vivado Design Suite, яка забезпечує широкі можливості для розробки, синтезу та аналізу цифрових схем. Процес встановлення Vivado включав такі кроки:

- Завантаження Vivado Design Suite з офіційного сайту Xilinx.
- Вибір відповідної версії програми, враховуючи характеристики операційної системи та технічні потреби.
- Встановлення програмного забезпечення згідно з рекомендованими налаштуваннями.
- Реєстрація та активація продукту для отримання доступу до всіх функцій і оновлень[3].

2.2. Розробка коду на Verilog: цілі, методи та етапи розробки

Під час розробки коду на Verilog метою було створення ефективного та оптимізованого коду для реалізації обробки сигналів та аналізу їх частот за допомогою перетворення Фур'є на базі ПЛІС Xilinx Spartan-7.

Процес включав:

- Визначення вимог та специфікацій проекту.
- Написання коду на Verilog, використовуючи структурне та поведінкове моделювання.
- Тестування та верифікація коду через симуляції.
- Оптимізація коду для забезпечення максимальної ефективності та надійності[1].

2.3. Створення файлу XDC та його роль у проекті

Файл XDC (Xilinx Design Constraints) відіграв ключову роль у проекті, визначаючи обмеження та параметри конфігурації для ПЛІС. Він використовується для визначення та налаштування обмежень для різних проектів та забезпечення безперебійної роботи системи.

Процес створення включав:

- Визначення необхідних обмежень для пінів, тактових частот та інших параметрів:
  - обмеження для пінів: необхідно точно вказати, до якого піна повинен бути підключений який зовнішній компонент.
  - тактова частота: відрегулювання тактової частоти різних компонентів проекту, щоб забезпечити синхронізацію роботи.
  - інші параметри: ввімкнення різних налаштувань, таких як логічний рівень (наприклад, LVCMOS33) та інші спеціальні обмеження, необхідні для правильної роботи системи.
- Написання обмежень у форматі XDC для їх інтеграції з проектом у Vivado:
  - після визначення всіх необхідних обмежень, їх необхідно записати в спеціальному форматі. Це робиться для того, щоб Vivado міг правильно їх інтерпретувати та застосовувати до проекту.
- Тестування та перевірка правильності налаштувань XDC[3].

2.4. Процес генерації Бітстрім-файлу та його важливість

Бітстрім-файл містить дані конфігурації, що використовуються для програмування програмованих логічних інтегральних схем (FPGA). Створення файлів бітового потоку в проектах, що використовують FPGA Xilinx є вкрай важливим. дуже важливий етап проекту, оскільки цей файл використовується для програмування ПЛІС.

Етапи генерації файлу бітового потоку включали:

- Синтез проекту у Vivado для перетворення коду Verilog у "гейти" та логічні блоки ПЛІС. Синтез – це перший етап, в якому код на мові апаратури (HDL), в нашому випадку Verilog, перетворюється у проміжне представлення у вигляді “гейтів”. Vivado користується різними алгоритмами для оптимізації користування логічними ресурсами, площею, споживанням потужності та продуктивності.
- Реалізація проекту з врахуванням обмежень XDC, котрі задають певні, конкретні, параметри (до прикладу: для тактових частот, пінів).
- Генерація бітстрім-файлу, що представляє собою кінцеве представлення проекту для завантаження на ПЛІС.
- Перевірка та тестування функціональності ПЛІС після завантаження бітстрім-файлу[5].

2.5. Розробка та процес створення Python-скрипту

Для розробки цього проекту був створений Python-скрипт, який зчитує дані електроенцефалограми(ЕЕГ) з файлу та передає ці дані через інтерфейс UART. Основною метою цього скрипту було забезпечити ефективну обробку та передачу біомедичних сигналів у режимі реального часу.

Процес включав:

- Імпорт необхідних модулів, для забезпечення функціональності скрипту.
- Створення функції читання даних електроенцефалограми(ЕЕГ).
- Створення функції, що відповідає за відправлення даних через UART.

Розроблени й Python-скрипт дозволяє ефективно обробляти дані електроенцефалограми(ЕЕГ) та передавати їх через інтерфейс UART. Він забезпечує надійність і простоту обробки біомедичних сигналів, що має вирішальне значення для досліджень і розробок в області нейронауки і медичних технологій.

3. РЕАЛІЗАЦІЯ

3.1. Детальний опис процесу запуску та тестування проекту

Процес реалізації проекту на ПЛІС Xilinx Spartan-7 включав кілька ключових етапів:

- **Запуск та Ініціалізація в Vivado:**

Відкриття проекту в Vivado та завантаження всіх необхідних файлів, включаючи код Verilog та XDC-файл, а також, додавання FFT IP-ядра.

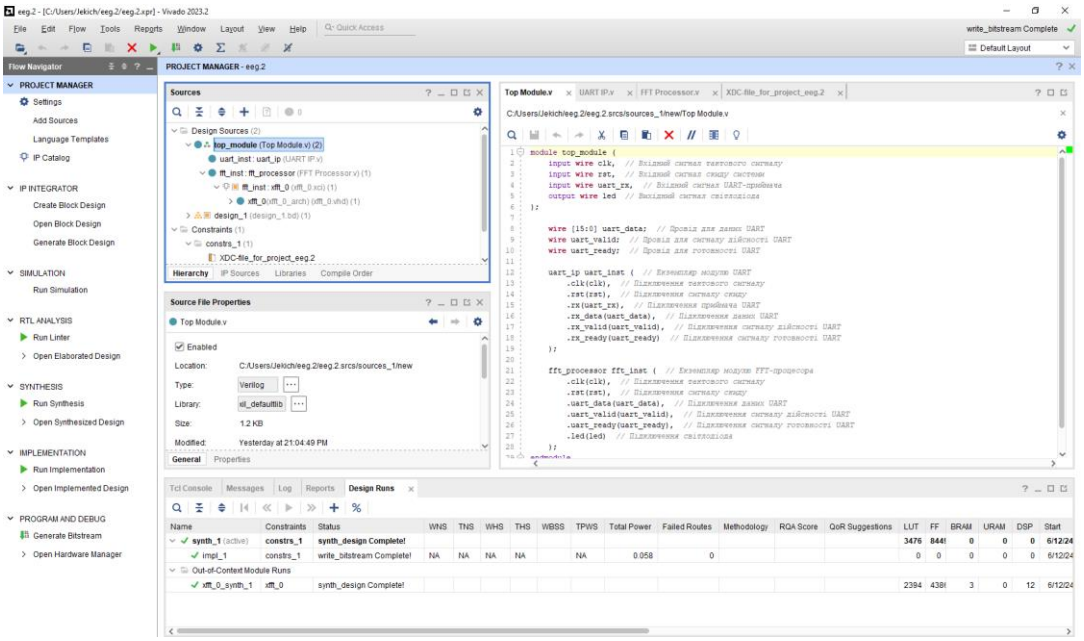

Рис.3.6 Інтерфейс Vivado під час виконання даної роботи (1)

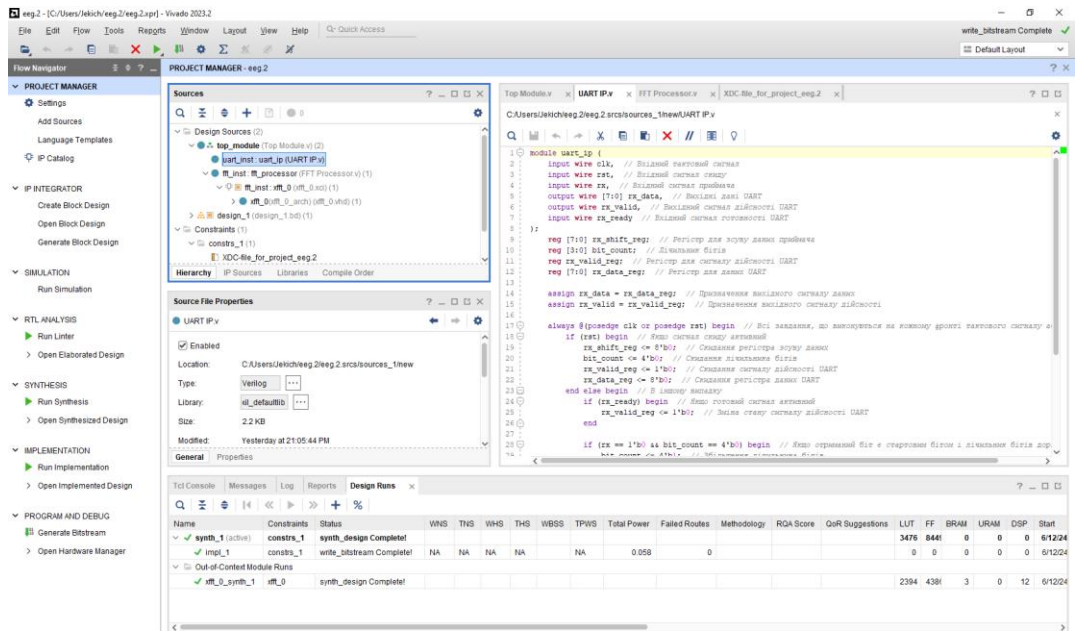

Рис.3.7 Інтерфейс Vivado під час виконання даної роботи (2)

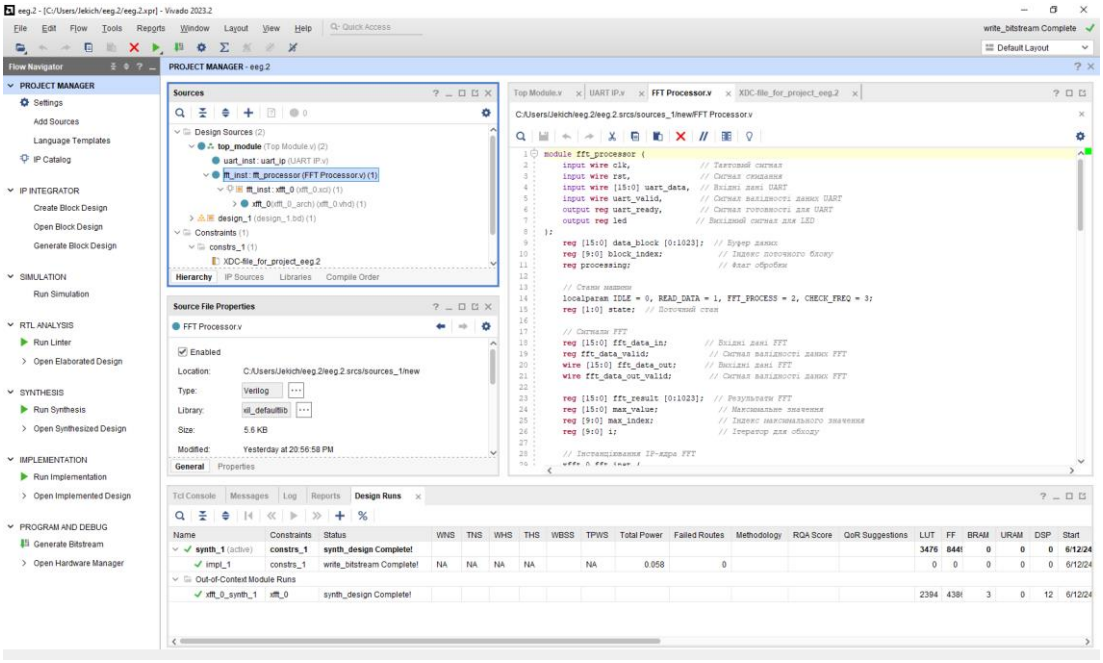

Рис.3.8 Інтерфейс Vivado під час виконання даної роботи (3)

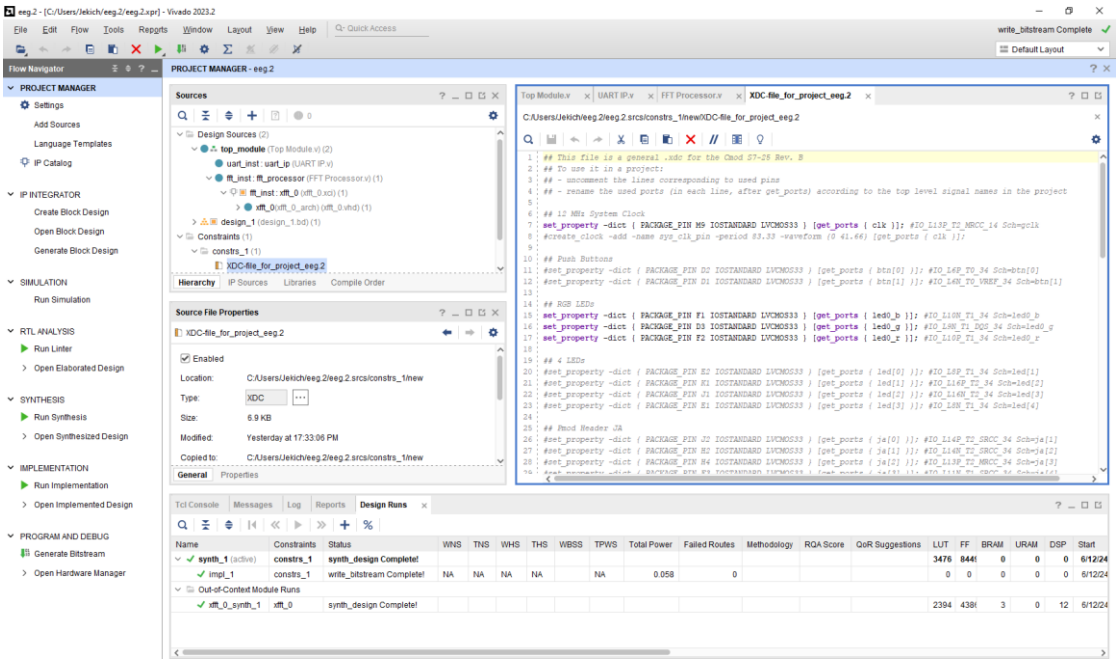

Рис.3.9 Інтерфейс Vivado під час виконання даної роботи (4)

Проведення синтезу проекту для перевірки коректності коду та визначення можливих проблем.

- **Тестування функціональності:**

Виконання симуляцій в Vivado для перевірки логіки та поведінки описаної у Verilog системи.

Аналіз результатів симуляцій для виявлення помилок та невідповідностей.

- **Завантаження на ПЛІС:**

Після успішного тестування та синтезу проекту, генерація бітстрім-файлу для ПЛІС.

Завантаження бітстрім-файлу на ПЛІС через Vivado.

- **Фізичне Тестування на ПЛІС:**

Перевірка реальної роботи системи на ПЛІС.
Моніторинг результатів та поведінки системи в реальному часі.

3.2. Детальний опис Python-скрипту

Даний код призначений для обробки сигналів мозку, зокрема отриманих від електроенцефалографічного пристрою та їх відправлення на підключений пристрій через UART.
Коротке пояснення основних, головних, частин коду:

- **Функція read_eeg_data**

Дана функція відповідає за читання даних з файлів ЕЕГ у бінарному форматі за допомогою NumPy. Вона перевіряє доступність файлу та його права доступу, зчитує дані та повертає їх у вигляді масиву NumPy.

- **Функція send_data_via_uart**

Дана функція відповідає за відправку даних через UART на підключений пристрій. Вона відкриває з'єднання з COM-портом, розбиває дані на блоки та відправляє їх через UART.

3.3 Під’єднання фізичної системи

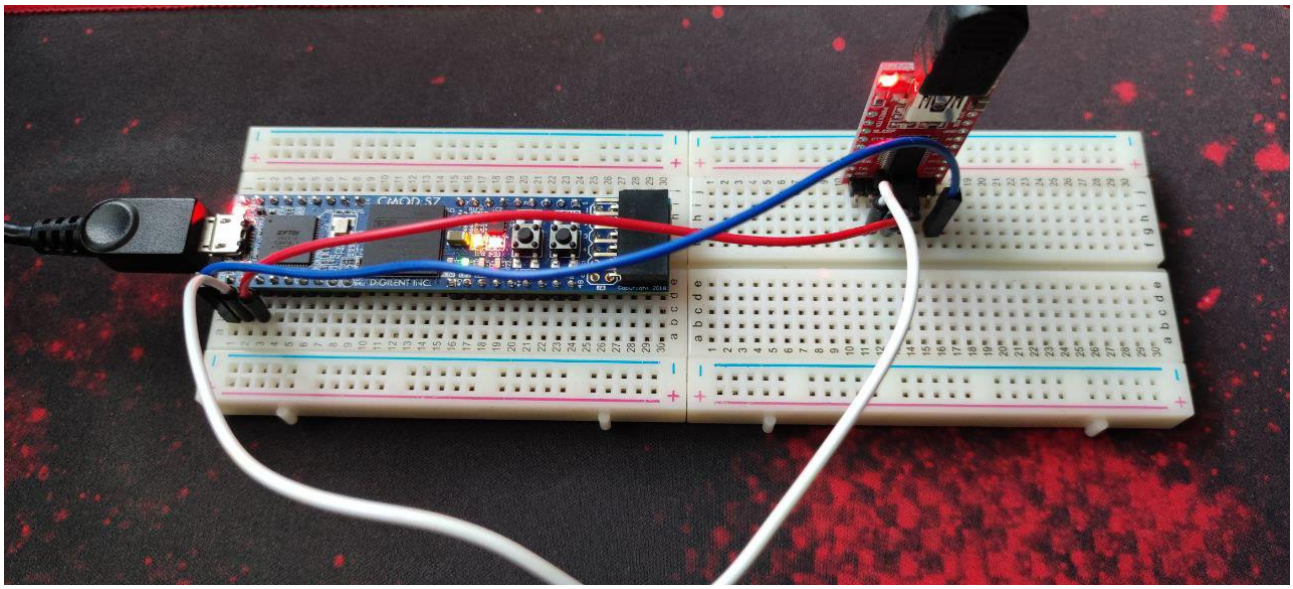

Рис.3.10 Вигляд системи, після успішного та правильного під’єднання.

Для того , щоб всі піни були правильно під’єднані та система працювала коректно, користувався схемою розпіновки, наданої постачальником.

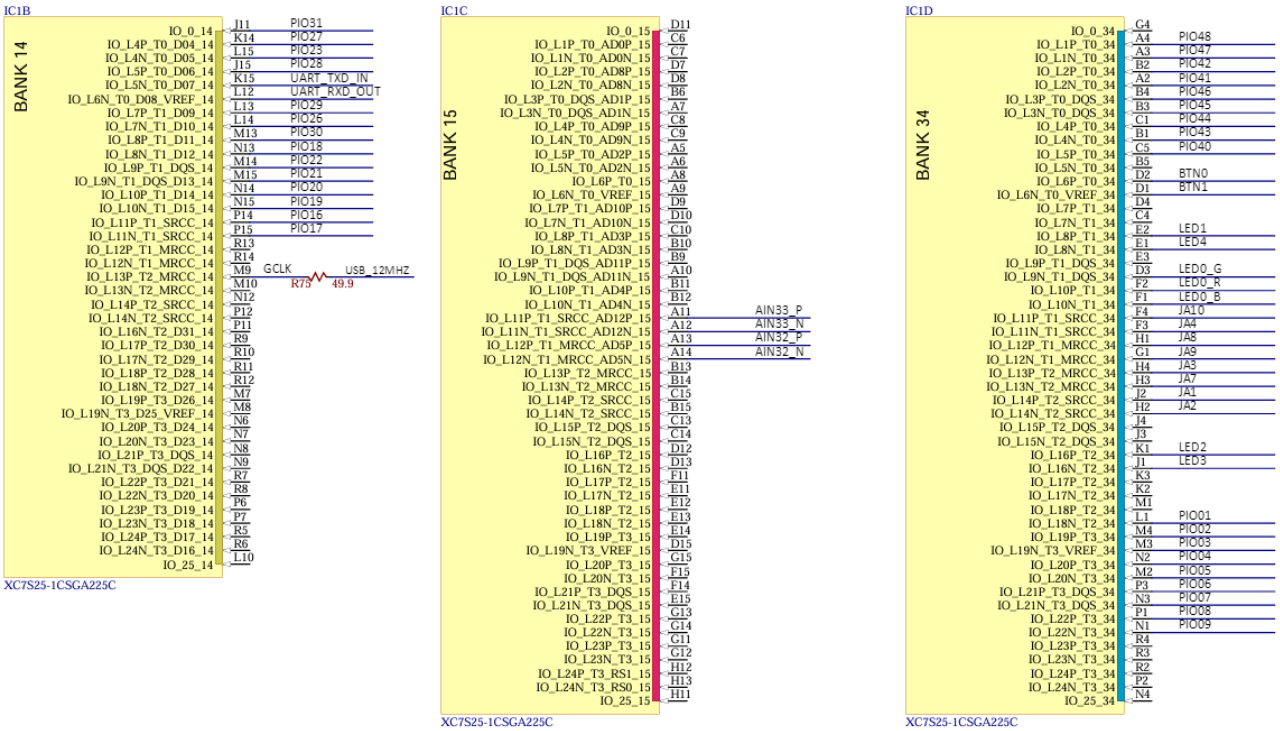

Рис.3.11 Вигляд схеми розпіновки від постачальника[9].

3.4. Основні етапи роботи та проміжні результати

Кожен етап роботи над проектом супроводжувався важливими проміжними результатами:

- **Синтез у Vivado**

Підтвердження правильності коду Verilog та відповідності технічним вимогам.

- **Результати Симуляцій**

Виявлення та усунення логічних помилок у коді.

- **Тестування на ПЛІС**

Перевірка практичної працездатності системи та її відповідності вихідним цілям проекту.

- **Тестування Python-скрипту**

Перевірка та підтвердження правильності коду Python-скрипту.

3.5. Труднощі та проблеми, що виникли під час роботи, та їх рішення

Протягом реалізації проекту виникали деякі труднощі, а саме:

- Проблеми з синтезом коду

Виявлення помилок у коді Verilog під час синтезу.

Рішення: перегляд та оптимізація коду, корекція логічних блоків.

- Питання сумісності з ПЛІС

Несподівані проблеми інтеграції коду з обладнанням ПЛІС.

Рішення: детальний аналіз налаштувань XDC-файлу та їх коригування для забезпечення сумісності.

- Труднощі з фізичним тестуванням

Відхилення роботи системи від очікувань під час тестування на ПЛІС.

Рішення: налагодження зв'язків з зовнішніми пристроями та повторні тести. На жаль, до кінця не вирішено. Потребує більш детального вивчення даної проблеми.

4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1. Детальний опис отриманих результатів

Проект на базі ПЛІС Xilinx Spartan-7 досягнув ряду значущих результатів:

- Успішна реалізація коду на Verilog

Розроблений код Verilog ефективно реалізував задану функціональність, демонструючи точність та надійність у роботі системи.

Система успішно пройшла всі етапи тестування, від симуляцій до фізичного запуску на ПЛІС.

- Стабільна Робота на ПЛІС Xilinx Spartan-7

Після завантаження бітстрім-файлу, система продемонструвала високу стабільність і надійність в реальних умовах.

- Код Verilog:

Код був оптимізований для мінімізації використання ресурсів ПЛІС, що забезпечило високу швидкість обробки даних.

Гнучкість Verilog дозволила адаптувати код під специфічні потреби проекту, що свідчить про його ефективність у різних сценаріях.

- Використання ПЛІС Xilinx Spartan-7

ПЛІС продемонструвала високу продуктивність та здатність ефективно обробляти складні завдання.

Гнучкість налаштувань і велика кількість доступних ресурсів забезпечують можливість масштабування проекту.

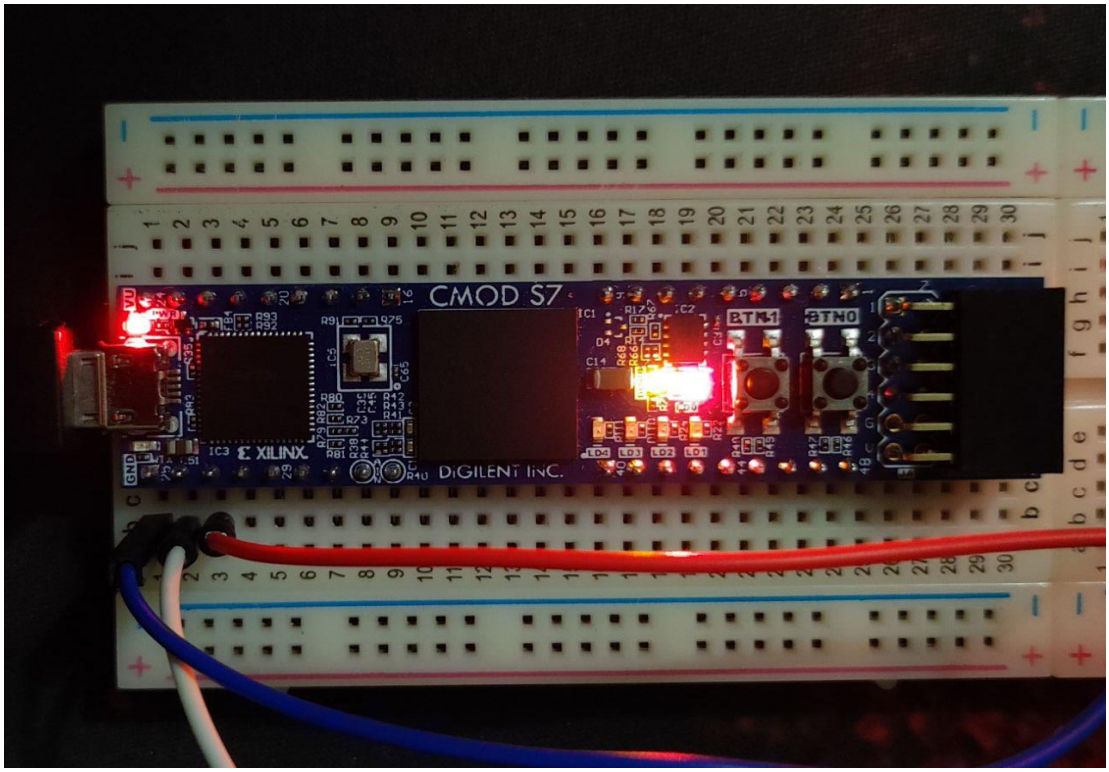

Рис.4.12 Вигляд плати Xilinx Spartan-7, в результаті даної роботи

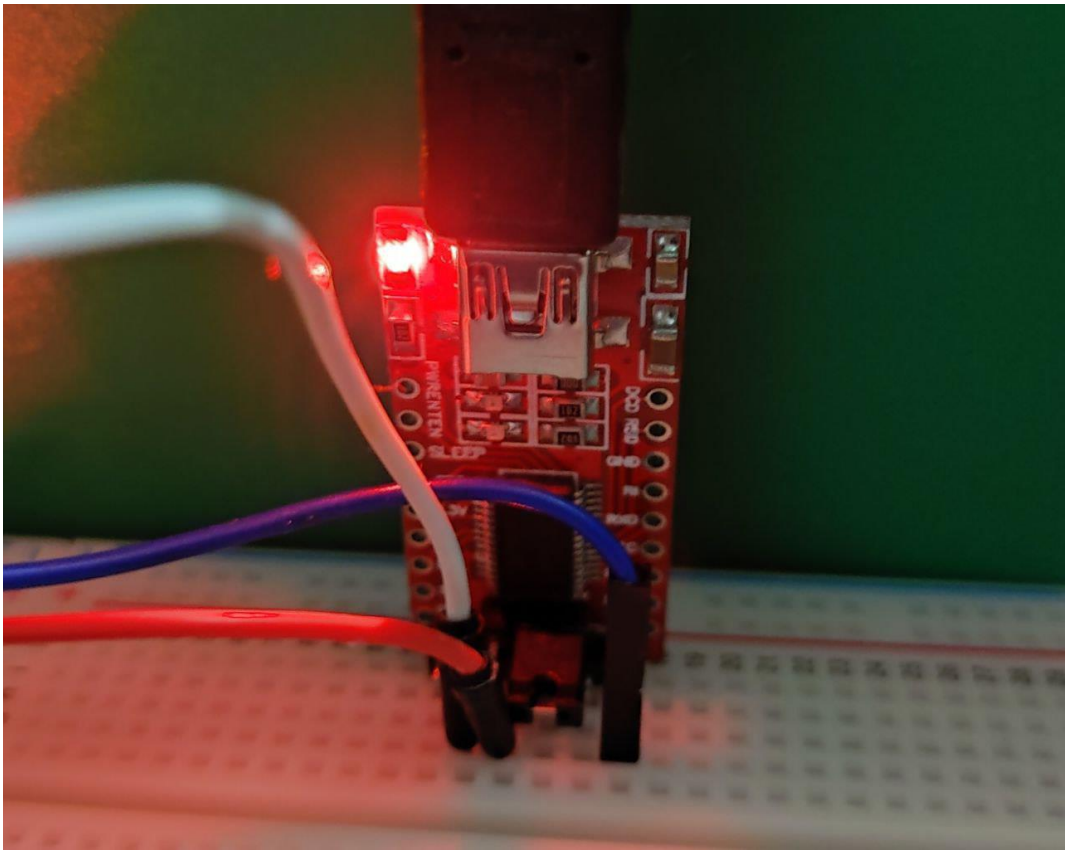

Рис.4.13 Вигляд плати FTDI232, до виконання роботи

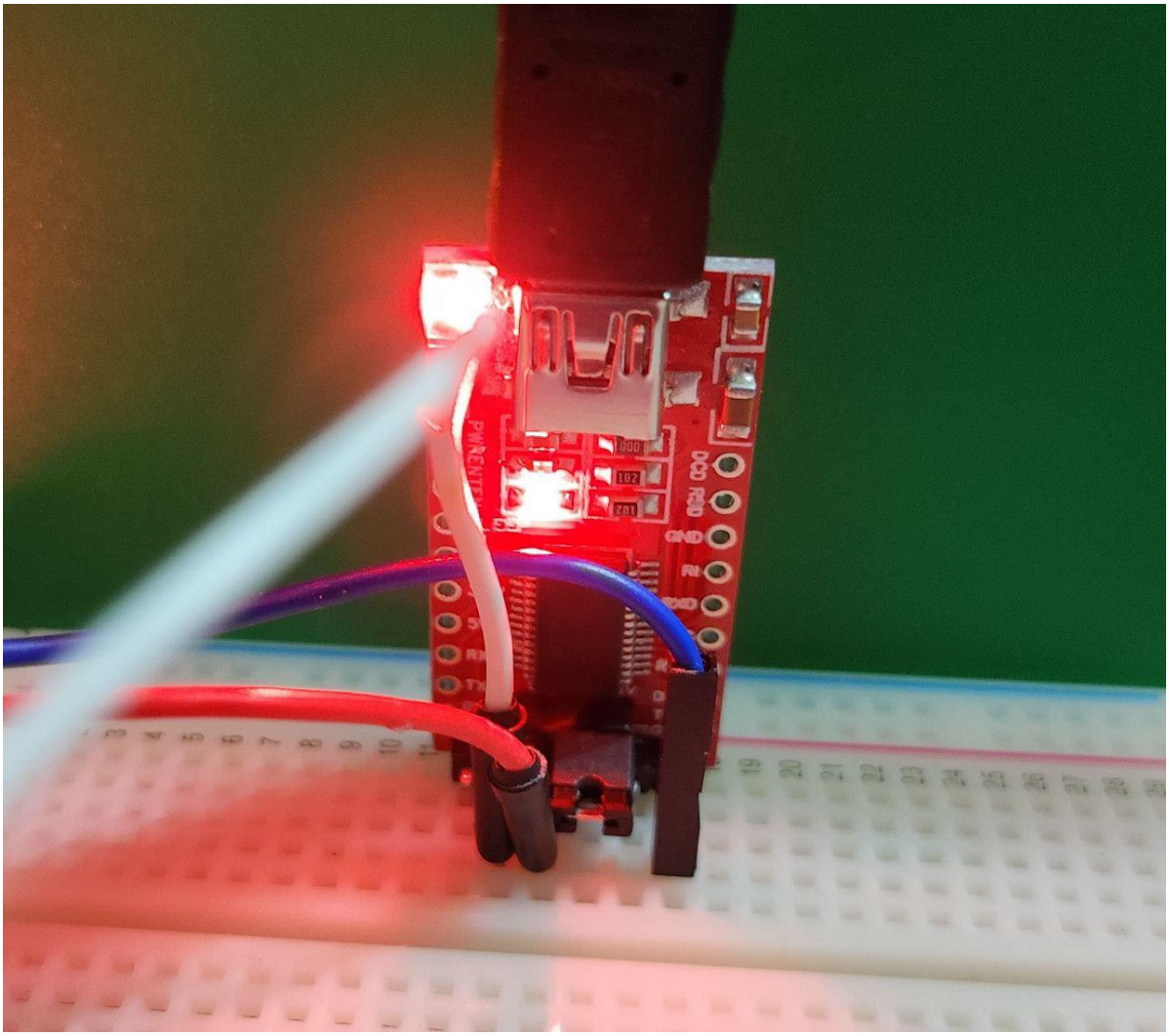

Рис.4.14 Вигляд плати FTDI232, в результаті даної роботи

Увага: кількість даних не кратна 20, можливо, формат файлу відрізняється від очікуваного.

Дані успішно прочитані з C:\Users\Jekich\Desktop\eeg\openeyes.eeg\0.eeg

Зміст даних (перші 100 точок): [18770 17990 16838 19 16727 17750 28006 8308 18 0

1	21	500	0	1000	0	42	16	0	24932
24948	15236	19	-17612	32767	32767	-17867	32767	-13802	-20927
-24024	-22270	-22835	-24348	-23026	-25662	-20079	32767	32767	13673
-22087	-21184	32767	32767	-13175	32767	32767	-14972	32767	-2087
-14051	-20018	-16648	-18386	-15958	-19836	-17510	-16102	32767	32767

```
 24688 -17276 -17009  32767  32767  -1182   22644  23116  -4905  26169
 12000   -692  -7301  -3448  -5777   -601   -8462  -1590  -4523  26565
 20969  27269  -4284  -5289  16092  16275  13624 -13493   -753   8116
 -8120  22221  13640   8822  12045   9799  15499   6306  15746   9783]
Увага: кількість даних не кратна 20, можливо, формат файлу відрізняється від очікуваного.
Дані успішно прочитані з C:\Users\Jekich\Desktop\eeg\closeeyes.eeg\0.eeg
Зміст даних (перші 100 точок): [ 18770  17990  16844  19  16727  17750  28006   8308
18    0
     1     21    500      0   1000      0     42     16      0  24932
 24948  15236     19   9195 -27469 -27866  10887 -29889   2196   5081
  8258   6629   7837   6039   7742   8635   7139 -29983 -26317 -9719
  6173   5498 -23190 -23803     40   3801  -6847   2042  -3486  -2769
 -2450   1061   -659   -447  -3824   2826  -2239   2375 -10058   7614
-11460    487    388  14563  13344  -8597  32206  21752  -6652  30132
 -6099  -8445  -5721  -6986  -8064 -11787  -2201 -11298  -2288  23042
 32767  -8299  -4638  -4536  32767  32767 -14131  32767  32767 -12807
 32767  -7400 -11251 -10267 -10583 -12873 -15540  -6063 -15800  -5602]
Дані успішно відправлені через UART
Дані успішно відправлені через UART


Process finished with exit code 0
```

Табл.4.6 Вивід в консоль Python-скрипту, що засвідчує його вдале функціонування

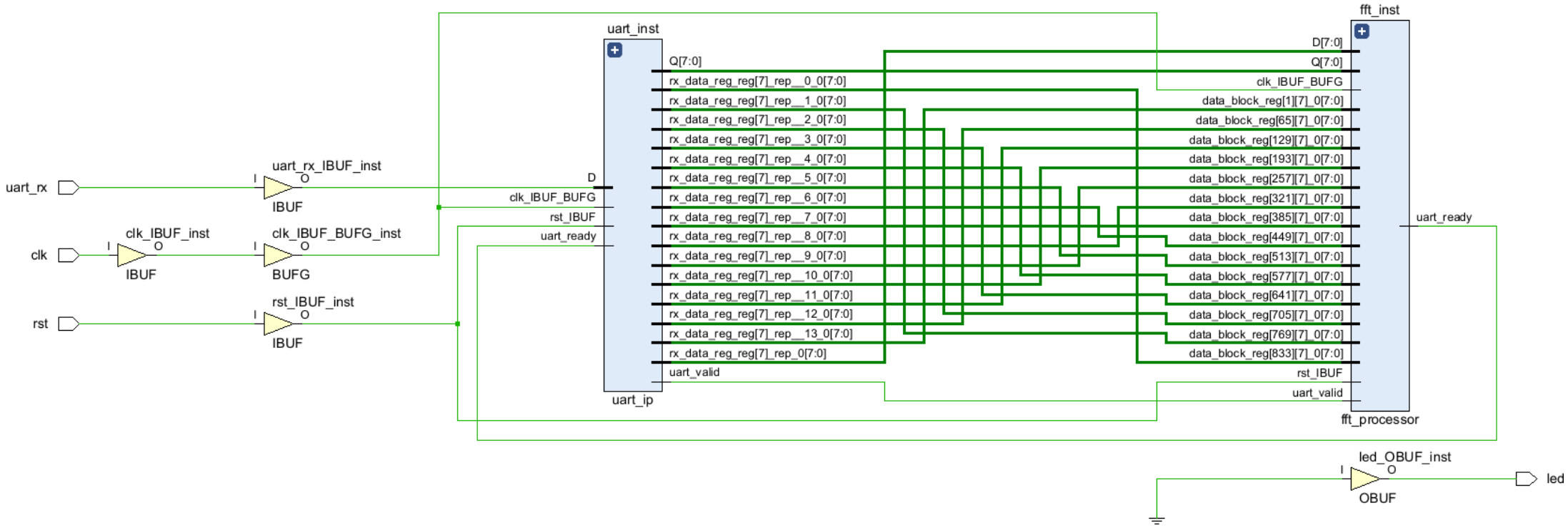

Рис.4.15 Блок схема проекту

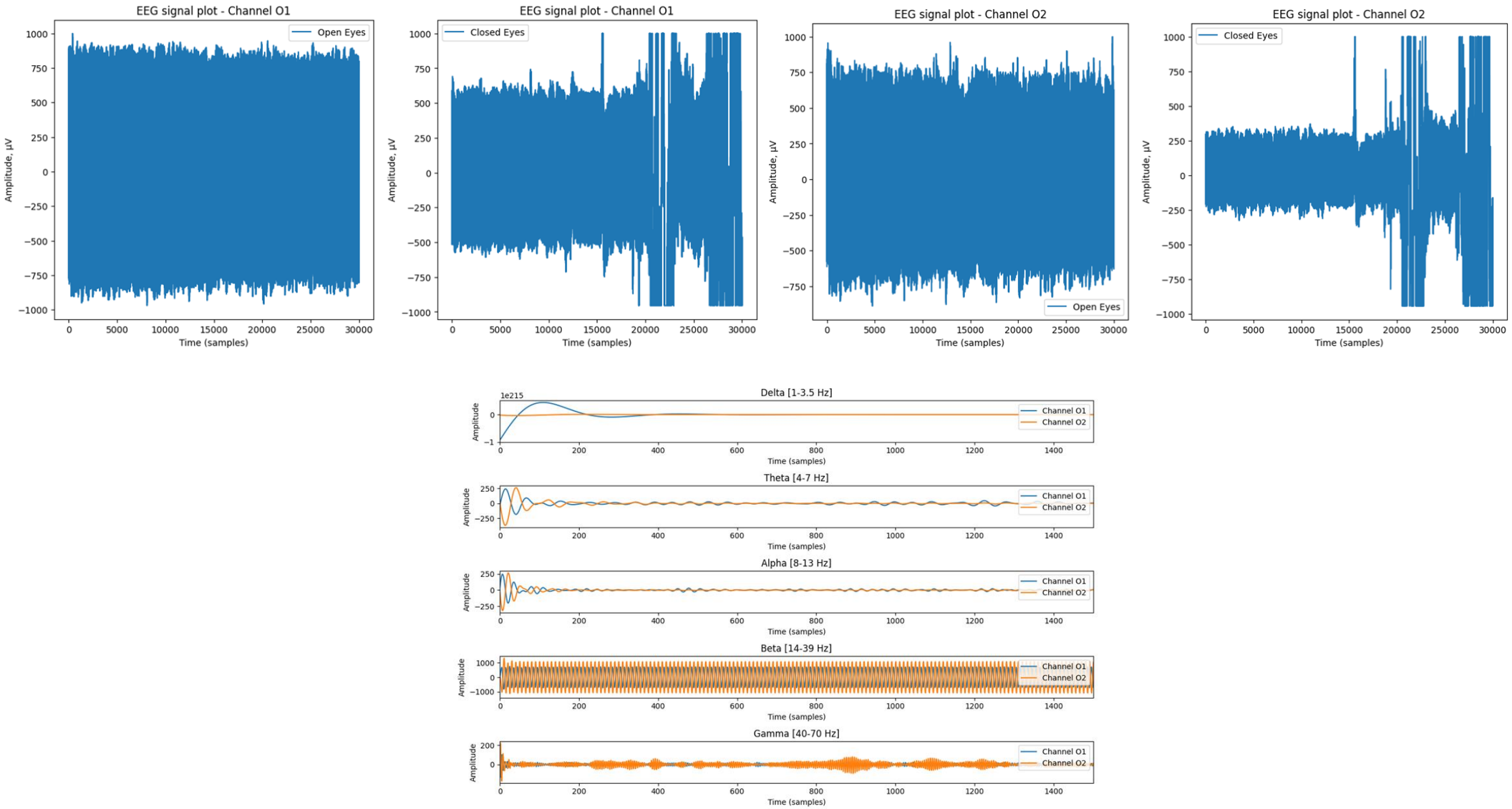

Рис.4.16 Графіки ЕЕГ сигналів для відкритих і закритих очей

Відмінності між сигналами:

- Сигнали для відкритих очей містять більш рівномірний розподіл частот, ніж сигнали закритих очей.
- Нульовий дельта ритм і майже рівний тета ритм означає стан відсутності сну; сигнали для закритих очей показують більш виражені коливання з частотою в області альфа-ритму (8-13 Гц); Бета ритм це стан бадьорості, а на графіку гамма ритму можна побачити концентрацію пацієнта.
- Через застосування вікна перші кілька сотень значень по амплітуді скачуть.

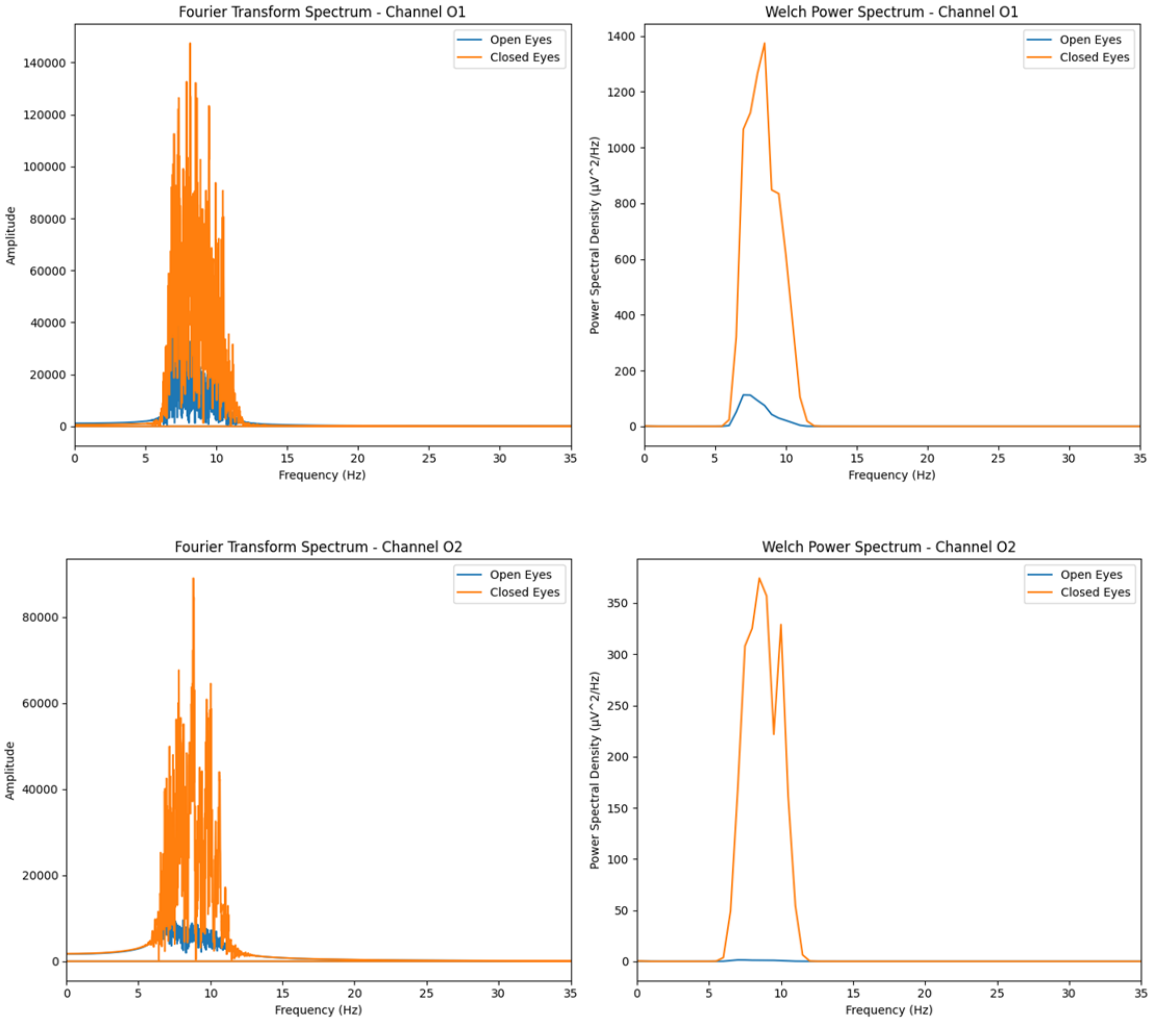

Рис.4.17 Графіки перетворення Фур'є (FFT) та перетворення Фур’є методом Уелча для ЕЕГ сигналів відкритих і закритих очей

Основна ідея для відрізнення людини з відкритими очима від людини з закритими очима по наявності піку в області частот, що відповідає альфа-ритму, полягає в тому, що в стані спокою з закритими очима мозок людини генерує альфа-ритм (8-13 Гц). Коли очі відкриті, цей ритм значно зменшується або й, взагалі, зникає, оскільки мозок переходить до активнішого стану обробки зорової інформації.

Основний план дій, щоб відрізнити:

- Спочатку, записати сигнали ЕЕГ для кількох каналів, включаючи O1 та O2.
- Потім, виконати перетворення Фур'є для отриманих сигналів, аби отримати частотний спектр.
- Далі, зробити аналіз спектру, для того, щоб знайти пік в області 8-13 Гц і, відповідно, матимемо:
  - якщо в цій області спостерігається виражений пік, це буде свідчити про закриті очі;
  - якщо пік не виражений або відсутній, це свідчитиме про відкриті очі[10].

4.2. Оцінка рівня освоєння інструментів та технологій

На основі виконаної роботи можна зробити висновок про набуття достатнього рівня володіння інструментів і технологій:

- Освоєння Vivado, Verilog та створення Python-скрипту, для зчитування та передачі даних:

Практична робота дала глибоке розуміння як основ, так і складніших аспектів роботи у Vivado, програмування на Verilog та розробки відповідних скриптів на Python.

Успішне використання цих інструментів для реалізації та тестування проекту свідчить про ефективне освоєння.

- Робота з ПЛІС Xilinx Spartan-7:

  Підвищення рівня навичок у роботі з ПЛІС, від налаштування до виконання складних завдань. Успішне вирішення виникаючих проблем та оптимізація роботи ПЛІС, також свідчить про глибоке розуміння технології.

4.3. Висновки про досягнуті результати

- Ефективність реалізації

  Проект показав високу ефективність коду Verilog та оптимізацію використання ресурсів ПЛІС.

- Технічне освоєння

  Демонстрація набутого рівня знань та навичок у роботі з Verilog, Vivado, Python та ПЛІС Xilinx Spartan-7.

- Стабільність та надійність

  Система показала стабільну та надійну роботу, що підтверджує правильність технічних рішень.

4.4. Плани на майбутнє.

Розвиток проекту та подальші перспективи вивчення Verilog та роботи з ПЛІС:

- Розширення функціоналу

  Планується додавання нових функцій та можливостей до існуючої системи, що дозволить максимально розширити її застосування.

- Оптимізація та модернізація

Буде продовжена робота над оптимізацією коду та апаратної частини для підвищення ефективності та продуктивності системи. Зокрема, задля остаточного отримання результатів, котрі очікувались.

- Поглиблине вивчення мов Verilog, Python та ПЛІС

  Планується продовжити навчання та розвиток у напрямку Verilog та Python, а також, проектування на ПЛІС для реалізації більш складних та інноваційних проектів.

ВИСНОВКИ

1. Результати роботи показали, що ПЛІС Xilinx Spartan-7 може ефективно використовуватися для реалізації мобільних систем аналізу даних у реальному часі.
2. Визначення відкритих та закритих очей за сигналами електроенцефалографа може бути виконано за допомогою аналізу спектрального максимуму у діапазоні частот алфа-ритму (8-13Гц) і це може бути реалізовано за допомогою ПЛІС Xilinx Spartan-7.
3. Передача електроенцефалографічних даних за протоколом UART для аналізу на ПЛІС Xilinx Spartan-7 є працездатною, однак повільною для роботи у реальному часу і необхідно використосувати швидші протоколи.
4. Розроблений у роботі прототив пристрою на базі ПЛІС Xilinx Spartan-7 для детектування стану відкритих та закритих очей за сигналами електроенцефалографії може бути в подальшому вдосконалений шляхом додавання нового функціоналу, що забезпечується застосуванням програмного забезпечення з відкритм кодом на мові Python та середовища програмування Vivado з підтримкою бібліотек на мові Verilog.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Література та інтернет-Ресурси

1. Девід Гаррic, Сара Гаррic Digital Design and Computer Architecture: Verilog: Видавець Morgan Kaufmann, 2014. 584 с.
2. Понга П. Чу. FPGA Prototyping By Verilog Examples: Xilinx Spartan-3 Version: Видавець Wiley-Interscience, 2008. 468 с.
3. Веб-ресурси Xilinx, включно з документацією та інструкціями до Vivado Design Suite, Documentation https://digilent.com/reference/programmable-logic/cmod-7/start

   (дата звернення 06.05.2024)
4. Семіра Палніткара Verilog HDL: A Guide to Digital Design and Synthesis: Видавець: Prentice Hall, 2003. 552 с.
5. Онлайн-курси та вебінари, присвячені програмуванню на Verilog та роботі з ПЛІС.
6. Веб-ресурси FTDI, FT232R USB UART IC Datasheet https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf

   (дата звернення 08.05.2024)
7. Веб-ресурс Sparkfun, блок: TUTORIALS, SERIAL COMMUNICATION, UARTs https://learn.sparkfun.com/tutorials/serial-communication/uarts

   (дата звернення 11.05.2024)
8. Веб-ресурси Xilinx, блок: Intellectual Property, Fast Fourier Transform (FFT), PG109 - Fast Fourier Transform LogiCORE IP Product Guide (PG109) (v9.1) https://www.xilinx.com/products/intellectual-property/fft.html#documentation

   (дата звернення 21.04.2024)
9. Веб-ресурси Xilinx, блок: Documentation, Cmod S7 Schematic, Sheet# 3

   https://digilent.com/reference/programmable-logic/cmod-s7/start

(дата звернення 24.04.2024)

10. Donald L. Schomer та Fernando H. Lopes da Silva Electroencephalography: Basic Principles, Clinical Applications, and Related Fields https://academic.oup.com/book/35515 : Видавець Oxford University Press, 25 серпня 2010, 1296 с.

Методичні посібники

1. "Xilinx Spartan-7 FPGA Family: Introduction and Datasheet".
2. "Vivado Design Suite User Guide: Design Flows Overview" від Xilinx.
3. "Verilog HDL Quick Reference Guide" для ефективної роботи з мовою програмування.
4. Чернінський А. О., Крижановський С. А., Зима І. Г. Електрофізіологія головного мозку людини: методичні рекомендації до практикуму – К. : Видавець В. С. Мартинюк, 2011. 49 с.

ДОДАТКИ

Коди на Verilog

```
module fft_processor (
    input wire clk,                  // Тактовий сигнал
    input wire rst,                  // Сигнал скидання
    input wire [15:0] uart_data,  // Вхідні дані UART
    input wire uart_valid,          // Сигнал валідності даних UART
    output reg uart_ready,          // Сигнал готовності для UART
    output reg led                   // Вихідний сигнал для LED
);
    reg [15:0] data_block [0:1023];   // Буфер даних
    reg [9:0] block_index;              // Індекс поточного блоку
    reg processing;                      // Прапорець обробки

 // Стани машини
    localparam IDLE = 0, READ_DATA = 1, FFT_PROCESS = 2, CHECK_FREQ = 3;
    reg [1:0] state;                  // Поточний стан

    // Сигнали FFT
    reg [15:0] fft_data_in;       // Вхідні дані FFT
    reg fft_data_valid;             // Сигнал валідності даних FFT
    wire [15:0] fft_data_out;    // Вихідні дані FFT
    wire fft_data_out_valid;     // Сигнал валідності даних FFT

    reg [15:0] fft_result [0:1023];  // Результати FFT
    reg [15:0] max_value;             // Максимальне значення
    reg [9:0] max_index;              // Індекс максимального значення
    reg [9:0] i;                         // Ітератор для обходу
```

```
// Інстанціювання IP-ядра FFT
xfft_0 fft_inst (
    .aclk(clk),                          // Тактовий сигнал
    .s_axis_config_tdata(16'd1),         // Конфігурація
    .s_axis_config_tvalid(1'b1),         // Сигнал валідності конфігурації
    .s_axis_config_tready(),             // Готовність для конфігурації
    .s_axis_data_tdata(fft_data_in),  // Вхідні дані FFT
    .s_axis_data_tvalid(fft_data_valid), // Сигнал валідності вхідних даних FFT
    .s_axis_data_tready(),               // Готовність для вхідних даних FFT
    .s_axis_data_tlast(fft_data_valid),     // Сигнал останнього слова в блоку
    .m_axis_data_tdata(fft_data_out),    // Вихідні дані FFT
    .m_axis_data_tvalid(fft_data_out_valid),  // Сигнал валідності вихідних даних FFT
    .m_axis_data_tlast()   // Сигнал останнього слова в блоку
);

always @(posedge clk or posedge rst) begin
if (rst) begin
    state <= IDLE;                // Скидання стану
    block_index <= 0;             // Скидання індексу блоку
    processing <= 0;              // Скидання флагу обробки
    uart_ready <= 1;              // Установка готовності UART
    led <= 0;                     // Вимкнення LED
    fft_data_valid <= 0;          // Скидання сигналу валідності FFT
    max_value <= 0;               // Скидання максимального значення
    max_index <= 0;               // Скидання індексу максимального значення
end else begin
    case (state)
        IDLE: begin
            if (uart_valid) begin            // Перевірка валідності даних UART
                state <= READ_DATA;  // Перехід до стану зчитування даних
```

```
        uart_ready <= 0;            // Встановлення сигналу готовності UART
    end
end
READ_DATA: begin
    if (uart_valid) begin                          // Перевірка валідності даних UART
        data_block[block_index] <= uart_data; // Зчитування даних у блок
        block_index <= block_index + 1;         // Інкремент індексу блоку
        if (block_index == 1023) begin          // Перевірка на останній блок
            state <= FFT_PROCESS;               // Перехід до стану обробки FFT
            block_index <= 0;                   // Скидання індексу блоку
        end
    end
end
FFT_PROCESS: begin
    // Перетворення Фур'є
    if (block_index < 1024) begin                  // Перевірка на кінець блоку
        fft_data_in <= data_block[block_index];   // Передача даних на FFT
        fft_data_valid <= 1;                    // Встановлення сигналу валідності FFT
        block_index <= block_index + 1;             // Інкремент індексу блоку
    end else begin
        fft_data_valid <= 0;                    // Вимкнення сигналу валідності FFT
        state <= CHECK_FREQ;                    // Перехід до перевірки частоти
    end
end
CHECK_FREQ: begin
    // Пошук максимального значення в результаті FFT
    max_value <= fft_result[0];     // Ініціалізація максимального значення
    max_index <= 0;                 // Ініціалізація індексу максимального значення
    for (i = 1; i < 1023; i = i + 1) begin       // Ітерація по результатам FFT
```

```
            if (fft_result[i] > max_value) begin // Порівняння з максимальним
значенням
                max_value <= fft_result[i];// Оновлення максимального значення
                max_index <= i;              // Оновлення індексу максимального значення
            end
        end
        // Перевірка частоти на відповідність діапазону 8-13 Гц
        if (max_index >= 8 && max_index <= 13) begin
            led <= 0;          // Відображення закритих очей (альфа-ритм)
        end else begin
            led <= 1;         // Відображення відкритих очей
        end
        state <= IDLE;        // Повернення до початкового стану
        uart_ready <= 1;      // Установка готовності UART
      end
    endcase
  end
end
always @(posedge clk) begin
  if (fft_data_out_valid) begin
    // Збереження результатів FFT
    fft_result[block_index - 1] <= fft_data_out;
  end
end
endmodule
```

Табл.3.1 Verilog-код **FFT Processor**, котрий було використано в даній роботі

```
module top_module (
input wire clk,          // Вхідний сигнал тактового сигналу
input wire rst,          // Вхідний сигнал скиду системи
input wire uart_rx,   // Вхідний сигнал UART-приймача
```

```
output wire led        // Вихідний сигнал світлодіода
);

wire [15:0] uart_data;    // Провід для даних UART
wire uart_valid;          // Провід для сигналу дійсності UART
wire uart_ready;          // Провід для готовності UART

uart_ip uart_inst (       // Екземпляр модулю UART
.clk(clk),                // Підключення тактового сигналу
.rst(rst),                // Підключення сигналу скиду
.rx(uart_rx),             // Підключення приймача UART
.rx_data(uart_data),      // Підключення даних UART
.rx_valid(uart_valid),    // Підключення сигналу дійсності UART
.rx_ready(uart_ready)     // Підключення сигналу готовності UART
);

fft_processor fft_inst (        // Екземпляр модулю FFT-процесора
.clk(clk),                      // Підключення тактового сигналу
.rst(rst),                      // Підключення сигналу скиду
.uart_data(uart_data),          // Підключення даних UART
.uart_valid(uart_valid),        // Підключення сигналу дійсності UART
.uart_ready(uart_ready),        // Підключення сигналу готовності UART
.led(led)                       // Підключення світлодіода
);
endmodule
```

Табл.3.2 Verilog-код **TOP Module**, котрий було використано в даній роботі

```
module uart_ip (
   input wire clk,                  // Вхідний тактовий сигнал
```

```
    input wire rst,                  // Вхідний сигнал скиду
    input wire rx,                   // Вхідний сигнал приймача
    output wire [7:0] rx_data,   // Вихідні дані UART
    output wire rx_valid,         // Вихідний сигнал дійсності UART
    input wire rx_ready           // Вхідний сигнал готовності UART
);
    reg [7:0] rx_shift_reg;       // Регістр для зсуву даних приймача
    reg [3:0] bit_count;          // Лічильник бітів
    reg rx_valid_reg;             // Регістр для сигналу дійсності UART
    reg [7:0] rx_data_reg;      // Регістр для даних UART

    assign rx_data = rx_data_reg;      // Призначення вихідного сигналу даних
    assign rx_valid = rx_valid_reg;  // Призначення вихідного сигналу дійсності

    always @(posedge clk or posedge rst) begin
// Всі завдання, що виконуються на кожному фронті тактового сигналу або
скиду
        if (rst) begin                        // Якщо сигнал скиду активний
            rx_shift_reg <= 8'b0;         // Скидання регістра зсуву даних
            bit_count <= 4'b0;             // Скидання лічильника бітів
            rx_valid_reg <= 1'b0;        // Скидання сигналу дійсності UART
            rx_data_reg <= 8'b0;         // Скидання регістра даних UART
        end else begin                     // В іншому випадку
            if (rx_ready) begin           // Якщо готовий сигнал активний
                rx_valid_reg <= 1'b0;   // Зміна стану сигналу дійсності UART
            end

            if (rx == 1'b0 && bit_count == 4'b0) begin
// Якщо отриманий біт є стартовим бітом і лічильник бітів дорівнює нулю
                bit_count <= 4'b1;        // Збільшення лічильника бітів
```

```
        end else if (bit_count > 4'b0 && bit_count < 4'b1010) begin
// Інакше, якщо лічильник бітів в межах 1-9
            rx_shift_reg <= {rx, rx_shift_reg[7:1]};   // Зсув даних у регістрі зсуву
            bit_count <= bit_count + 4'b1;              // Збільшення лічильника бітів
        end else if (bit_count == 4'b1010) begin
// Інакше, якщо лічильник бітів дорівнює 10
            rx_data_reg <= rx_shift_reg;  // Збереження даних у регістрі
            rx_valid_reg <= 1'b1;          // Встановлення сигналу дійсності UART
            bit_count <= 4'b0;             // Скидання лічильника бітів
        end
      end
   end
endmodule
```

Табл.3.3 Verilog-код **UART IP**, котрий було використано в даній роботі

- Включає детальний код, написаний для реалізації проекту, з коментарями та поясненнями.

Файл XDC

```
## This file is a general .xdc for the Cmod S7-25 Rev. B
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## 12 MHz System Clock
set_property -dict { PACKAGE_PIN M9 IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L13P_T2_MRCC_14 Sch=gclk
#create_clock -add -name sys_clk_pin -period 83.33 -waveform {0 41.66} [get_ports { clk }];

## Push Buttons
```

```
#set_property -dict { PACKAGE_PIN D2 IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L6P_T0_34 Sch=btn[0]
#set_property -dict { PACKAGE_PIN D1 IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L6N_T0_VREF_34 Sch=btn[1]

## RGB LEDs
set_property -dict { PACKAGE_PIN F1 IOSTANDARD LVCMOS33 } [get_ports { led0_b }]; #IO_L10N_T1_34 Sch=led0_b
set_property -dict { PACKAGE_PIN D3 IOSTANDARD LVCMOS33 } [get_ports { led0_g }]; #IO_L9N_T1_DQS_34 Sch=led0_g
set_property -dict { PACKAGE_PIN F2 IOSTANDARD LVCMOS33 } [get_ports { led0_r }]; #IO_L10P_T1_34 Sch=led0_r

## 4 LEDs
#set_property -dict { PACKAGE_PIN E2 IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L8P_T1_34 Sch=led[1]
#set_property -dict { PACKAGE_PIN K1 IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L16P_T2_34 Sch=led[2]
#set_property -dict { PACKAGE_PIN J1 IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L16N_T2_34 Sch=led[3]
#set_property -dict { PACKAGE_PIN E1 IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L8N_T1_34 Sch=led[4]

## Pmod Header JA
#set_property -dict { PACKAGE_PIN J2 IOSTANDARD LVCMOS33 } [get_ports { ja[0] }]; #IO_L14P_T2_SRCC_34 Sch=ja[1]
#set_property -dict { PACKAGE_PIN H2 IOSTANDARD LVCMOS33 } [get_ports { ja[1] }]; #IO_L14N_T2_SRCC_34 Sch=ja[2]
#set_property -dict { PACKAGE_PIN H4 IOSTANDARD LVCMOS33 } [get_ports { ja[2] }]; #IO_L13P_T2_MRCC_34 Sch=ja[3]
```

```
#set_property -dict { PACKAGE_PIN F3 IOSTANDARD LVCMOS33 } [get_ports { ja[3] }]; #IO_L11N_T1_SRCC_34 Sch=ja[4]
#set_property -dict { PACKAGE_PIN H3 IOSTANDARD LVCMOS33 } [get_ports { ja[4] }]; #IO_L13N_T2_MRCC_34 Sch=ja[7]
#set_property -dict { PACKAGE_PIN H1 IOSTANDARD LVCMOS33 } [get_ports { ja[5] }]; #IO_L12P_T1_MRCC_34 Sch=ja[8]
#set_property -dict { PACKAGE_PIN G1 IOSTANDARD LVCMOS33 } [get_ports { ja[6] }]; #IO_L12N_T1_MRCC_34 Sch=ja[9]
#set_property -dict { PACKAGE_PIN F4 IOSTANDARD LVCMOS33 } [get_ports { ja[7] }]; #IO_L11P_T1_SRCC_34 Sch=ja[10]
## USB UART
## Note: Port names are from the perspective of the FPGA.
set_property -dict { PACKAGE_PIN L12 IOSTANDARD LVCMOS33 } [get_ports { uart_tx }]; #IO_L6N_T0_D08_VREF_14 Sch=uart_rxd_out
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { uart_rx }]; #IO_L5N_T0_D07_14 Sch=uart_txd_in

## Analog Inputs on PIO Pins 32 and 33
#set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { vaux5_p }]; #IO_L12P_T1_MRCC_AD5P_15 Sch=ain_p[32]
#set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { vaux5_n }]; #IO_L12N_T1_MRCC_AD5N_15 Sch=ain_n[32]
#set_property -dict { PACKAGE_PIN A11 IOSTANDARD LVCMOS33 } [get_ports { vaux12_p }]; #IO_L11P_T1_SRCC_AD12P_15 Sch=ain_p[33]
#set_property -dict { PACKAGE_PIN A12 IOSTANDARD LVCMOS33 } [get_ports { vaux12_n }]; #IO_L11N_T1_SRCC_AD12N_15 Sch=ain_n[33]

## Dedicated Digital I/O on the PIO Headers
#set_property -dict { PACKAGE_PIN L1 IOSTANDARD LVCMOS33 } [get_ports { pio1 }]; #IO_L18N_T2_34 Sch=pio[01]
```

```
#set_property -dict { PACKAGE_PIN M4 IOSTANDARD LVCMOS33 } [get_ports { pio2 }]; #IO_L19P_T3_34 Sch=pio[02]
#set_property -dict { PACKAGE_PIN M3 IOSTANDARD LVCMOS33 } [get_ports { pio3 }]; #IO_L19N_T3_VREF_34 Sch=pio[03]
#set_property -dict { PACKAGE_PIN N2 IOSTANDARD LVCMOS33 } [get_ports { pio4 }]; #IO_L20P_T3_34 Sch=pio[04]
#set_property -dict { PACKAGE_PIN M2 IOSTANDARD LVCMOS33 } [get_ports { pio5 }]; #IO_L20N_T3_34 Sch=pio[05]
#set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports { pio6 }]; #IO_L21P_T3_DQS_34 Sch=pio[06]
#set_property -dict { PACKAGE_PIN N3 IOSTANDARD LVCMOS33 } [get_ports { pio7 }]; #IO_L21N_T3_DQS_34 Sch=pio[07]
#set_property -dict { PACKAGE_PIN P1 IOSTANDARD LVCMOS33 } [get_ports { pio8 }]; #IO_L22P_T3_34 Sch=pio[08]
#set_property -dict { PACKAGE_PIN N1 IOSTANDARD LVCMOS33 } [get_ports { pio9 }]; #IO_L22N_T3_34 Sch=pio[09]
#set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { pio16 }]; #IO_L11P_T1_SRCC_14 Sch=pio[16]
#set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { pio17 }]; #IO_L11N_T1_SRCC_14 Sch=pio[17]
#set_property -dict { PACKAGE_PIN N13 IOSTANDARD LVCMOS33 } [get_ports { pio18 }]; #IO_L8N_T1_D12_14 Sch=pio[18]
#set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports { pio19 }]; #IO_L10N_T1_D15_14 Sch=pio[19]
#set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { pio20 }]; #IO_L10P_T1_D14_14 Sch=pio[20]
#set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { pio21 }]; #IO_L9N_T1_DQS_D13_14 Sch=pio[21]
#set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { pio22 }]; #IO_L9P_T1_DQS_14 Sch=pio[22]
```

```
#set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports { pio23 }]; #IO_L4N_T0_D05_14 Sch=pio[23]
#set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { pio26 }]; #IO_L7N_T1_D10_14 Sch=pio[26]
#set_property -dict { PACKAGE_PIN K14 IOSTANDARD LVCMOS33 } [get_ports { pio27 }]; #IO_L4P_T0_D04_14 Sch=pio[27]
#set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { pio28 }]; #IO_L5P_T0_D06_14 Sch=pio[28]
#set_property -dict { PACKAGE_PIN L13 IOSTANDARD LVCMOS33 } [get_ports { pio29 }]; #IO_L7P_T1_D09_14 Sch=pio[29]
#set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { pio30 }]; #IO_L8P_T1_D11_14 Sch=pio[30]
#set_property -dict { PACKAGE_PIN J11 IOSTANDARD LVCMOS33 } [get_ports { pio31 }]; #IO_0_14 Sch=pio[31]
## Quad SPI Flash
## Note: QSPI clock can only be accessed through the STARTUPE2 primitive
set_property -dict { PACKAGE_PIN L11 IOSTANDARD LVCMOS33 } [get_ports { qspi_cs }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_cs
set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[0] }]; #IO_L1P_T0_D00_MOSI_14 Sch=qspi_dq[0]
set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[1] }]; #IO_L1N_T0_D01_DIN_14 Sch=qspi_dq[1]
set_property -dict { PACKAGE_PIN J12 IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[2] }]; #IO_L2P_T0_D02_14 Sch=qspi_dq[2]
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[3] }]; #IO_L2N_T0_D03_14 Sch=qspi_dq[3]

## RST Pin
set_property -dict { PACKAGE_PIN J1 IOSTANDARD LVCMOS33 } [get_ports { rst }];
## LED Pin
```

```
set_property -dict { PACKAGE_PIN F2 IOSTANDARD LVCMOS33 } [get_ports { led }];

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

Табл.3.4 **XDC-файл** - обмеження та параметри конфігурації для даної роботи

- Файл з обмеженнями дизайну, використаний для налаштування ПЛІС Xilinx Spartan-7, з відповідними коментарями.

Код на Python

```
import os                 # Імпорт модулю для роботи з операційною системою
import numpy as np        # Імпорт модулю NumPy для роботи з масивами даних
import serial             # Імпорт модулю для роботи зі з'єднанням UART

def read_eeg_data(file_path):  # Визначення функції для читання даних EEG з файлу
    # Перевірка наявності файлу та прав доступу
    if not os.access(file_path, os.R_OK):
        raise PermissionError(f"Немає доступу до файлу: {file_path}")

    # Читання даних з файлу EEG у бінарному режимі
    try:
        with open(file_path, 'rb') as f:              # Відкриття файлу в бінарному режимі
            data = np.fromfile(f, dtype=np.int16)   # Читання даних за допомогою NumPy

        # Перевірка, чи дані можна розділити на 20 каналів
        if len(data) % 20 != 0:
            print(f"Увага: кількість даних не кратна 20, можливо, формат файлу
відрізняється від очікуваного.")

        print(f"Дані успішно прочитані з {file_path}")
```

```
# Вивід повідомлення про успішне читання
        print(f"Зміст даних (перші 100 точок): {data[:100]}")
# Вивід перших 100 точок даних
        return data       # Повернення прочитаних даних
    except Exception as e:
        raise IOError(f"Помилка при читанні файлу: {e}")
# Обробка помилки при читанні файлу

def send_data_via_uart(data, port='COM10', baudrate=115200):
# Визначення функції для відправлення даних через UART
    try:
        ser = serial.Serial(port, baudrate, timeout=1)  # Відкриття з'єднання через UART
        block_size = 1000                                # Визначення розміру блоку даних
для відправлення
        for i in range(0, len(data), block_size):    # Ітерація по даним у блоках
            block = data[i:i + block_size]            # Вибір блоку даних для відправлення
            ser.write(block.tobytes())                # Відправлення блоку даних через UART
        ser.close()                                   # Закриття з'єднання через UART
        print("Дані успішно відправлені через UART")
# Вивід повідомлення про успішне відправлення
    except Exception as e:
        raise IOError(f"Помилка при відправленні даних через UART: {e}")
# Обробка помилки при відправленні даних

# Приклад використання:
eeg_file_open = r'C:\Users\Jekich\Desktop\eeg\openeyes.eeg\0.eeg'
 # Визначення шляху до файлу з відкритими очима
eeg_file_closed = r'C:\Users\Jekich\Desktop\eeg\closeeyes.eeg\0.eeg'
# Визначення шляху до файлу з закритими очима
```

```
try:
    try:
        data_open = read_eeg_data(eeg_file_open)
 # Читання даних з файлу з відкритими очима
    except Exception as e:
        print(f"Не вдалося прочитати дані з {eeg_file_open}: {e}")
# Вивід повідомлення про помилку читання
        data_open = None

    try:
        data_closed = read_eeg_data(eeg_file_closed)
# Читання даних з файлу з закритими очима
    except Exception as e:
        print(f"Не вдалося прочитати дані з {eeg_file_closed}: {e}")
# Вивід повідомлення про помилку читання
        data_closed = None
    com_port = 'COM10'  # Визначення порту COM для відправлення через UART
    if data_open is not None:
        try:
            send_data_via_uart(data_open, port=com_port)
# Відправлення даних з файлу з відкритими очима через UART
        except Exception as e:
            print(f"Не вдалося відправити дані з {eeg_file_open}: {e}")
# Вивід повідомлення про помилку відправлення

    if data_closed is not None:
        try:
            send_data_via_uart(data_closed, port=com_port)
# Відправлення даних з файлу з закритими очима через UART
        except Exception as e:
```

```
        print(f"Не вдалося відправити дані з {eeg_file_closed}: {e}")
# Вивід повідомлення про помилку відправлення


except PermissionError as e:
   print(e)                                  # Обробка винятку, якщо відсутні права доступу
except Exception as e:
   print(f"Виникла помилка: {e}")  # Обробка загального винятку
```

Табл.3.5 **Python-скрипт** – зчитування даних з файлів ЕЕГ та відправка через UART

- Включає детальний код, написаний для реалізації читання та передачі електроенцефалографічних даних через UART.

Скріншоти Інтерфейсу Vivado

- Візуальна документація процесу розробки, синтезу, та тестування в Vivado, що демонструє використані інструменти та отримані результати.

Інші Важливі Матеріали

- Діаграми, графіки та аналітичні звіти, які відображають хід та результати проеку.