

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

Кафедра теории вероятностей и анализа данных

Направление подготовки: «Фундаментальная информатика и
информационные технологии»

Профиль подготовки: «Инженерия программного обеспечения»

**Отчёт по
учебной научно-исследовательской работе**

на тему:

**«Разработка аппаратно-программного комплекса для
получения ЭЭГ»**

Выполнил: студент группы 3822Б1ФИ1

Чистов А.Д.

Подпись

Научный руководитель:

доцент,

кандидат технических наук

Борисов Н.А.

Подпись

Нижний Новгород
2025

Содержание

Введение	3
1. ESP-32.....	4
2. Модуль AD8232.....	7
3. Программно-аппаратный комплекс	10
4. Neurosky Mobile.....	12
4.1 Мозговые волны и их значение в электроэнцефалографии	15
4.2 Протокол передачи данных устройства NeuroSky MindWave Mobile	17
4.3 Реализация и тестирование приёма данных от устройства MindWave	19
Список литературы	21
Приложения.....	22
Приложение А. Реализация класса Device.....	22
Приложение Б. Реализация класса Form	23
Приложение В. Реализация класса Filter	28
Приложение Г. Реализация класса MindWaveReader.....	29

Введение

Исследование электроэнцефалограммы (ЭЭГ) представляет собой междисциплинарное направление, объединяющее такие области знаний, как нейробиология, медицина и технологии обработки данных. Главной задачей этого направления является изучение активности головного мозга для решения широкого спектра задач, включая диагностику заболеваний, реабилитацию пациентов и создание инновационных интерфейсов мозг-компьютер.

Процесс получения и анализа ЭЭГ является весьма сложным и требует учета множества факторов. Работа начинается с использования высокочувствительных датчиков, которые регистрируют электрическую активность мозга через кожу головы. Эти сигналы затем усиливаются, фильтруются и обрабатываются с помощью специализированного программного обеспечения. На каждом этапе необходимо учитывать как физиологические особенности человека, так и технические аспекты работы оборудования, чтобы обеспечить точность и надежность результатов.

Разработка программно-аппаратного комплекса для работы с ЭЭГ включает:

1. **Тестирование и получение данных с существующих ЭЭГ-приборов.** Проверка точности, качества сигналов и выявление слабых мест современных систем.
2. **Изучение принципов и устройств считывания биологического сигнала с кожи человека.** Изучение характеристик электродов, методов их применения и их влияния на точность измерений.
3. **Разработка ПО для получения и визуализации сигнала ЭЭГ.** Создание программных инструментов для обработки, анализа и удобной интерпретации данных.

Результаты работы по данным направлениям позволяют создать основу для дальнейших исследований, направленных на улучшение точности диагностики, повышение доступности ЭЭГ-систем и их применения в реальном времени.

1. ESP-32

ESP-32 — это современная высокопроизводительная микроконтроллерная плата, разработанная компанией Espressif Systems, широко применяемая в области интернета вещей (IoT), автоматизации и управления. Благодаря своим техническим характеристикам и функциональным возможностям, плата ESP-32 (Рис. 1). зарекомендовала себя как универсальное решение для множества инженерных и исследовательских задач.

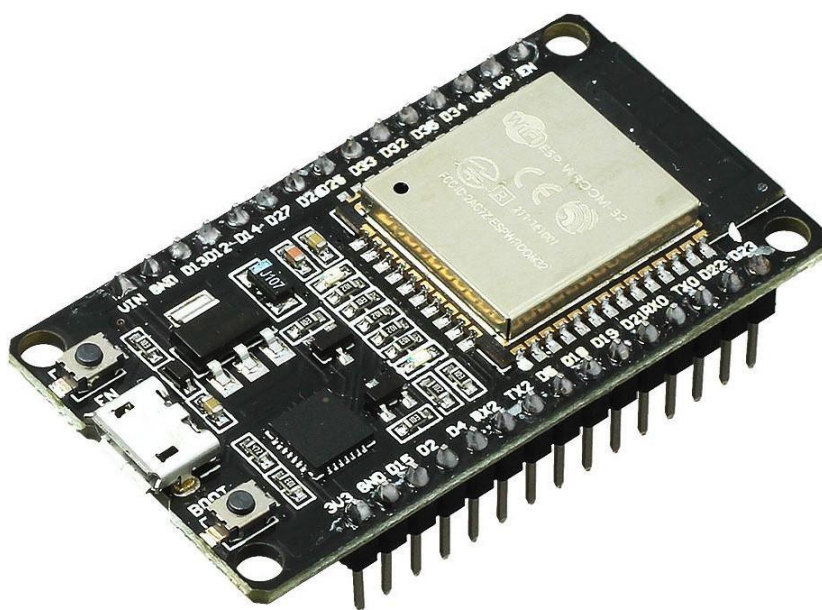


Рис. 1. Отладочная плата ESP-32

Технические характеристики ESP-32:

1. **Процессор:**
 - Двухъядерный Tensilica LX6 с тактовой частотой до 240 МГц, обеспечивающий высокую производительность для выполнения сложных вычислительных операций.
2. **Коммуникационные возможности:**
 - Поддержка Wi-Fi (стандарт 802.11 b/g/n) и Bluetooth 4.2, включая BLE (Bluetooth Low Energy). Это делает ESP-32 идеальным выбором для задач, связанных с беспроводной передачей данных.
3. **Память:**
 - Встроенная оперативная память (520 КБ SRAM) и флеш-память (до 4 МБ), что позволяет обрабатывать значительные объёмы данных и хранить программы большого размера.
4. **Интерфейсы и порты ввода-вывода:**
 - Поддержка GPIO, UART, SPI, I2C, PWM, ADC, DAC и других интерфейсов, что обеспечивает лёгкую интеграцию с датчиками, исполнительными механизмами и внешними устройствами.
5. **Энергопотребление:**

- Возможность работы в различных режимах энергосбережения, таких как глубокий сон (Deep Sleep), позволяет применять ESP-32 в автономных устройствах с ограниченным питанием.

Преимущества ESP-32 по сравнению с Arduino:

- 1. Беспроводная связь:**
 - ESP-32 имеет встроенные модули Wi-Fi и Bluetooth, в то время как в платах Arduino, например, Arduino Uno, такие функции отсутствуют и требуют дополнительных модулей.
- 2. Высокая производительность:**
 - Двухъядерный процессор и увеличенные объёмы памяти дают значительное преимущество в обработке данных по сравнению с 8-битными микроконтроллерами, такими как ATmega328P в Arduino Uno.
- 3. Многозадачность:**
 - Поддержка RTOS (например, FreeRTOS) позволяет организовать параллельное выполнение нескольких задач, что существенно упрощает разработку сложных систем.
- 4. Стоимость:**
 - Несмотря на расширенный функционал, ESP-32 остаётся доступным решением и часто дешевле плат Arduino с эквивалентным функционалом.
- 5. Широкий спектр приложений:**
 - ESP-32 используется в IoT, управлении роботами, системах мониторинга окружающей среды и других приложениях, где Arduino может быть ограничен по своим возможностям.

Практическое применение ESP-32 на начальных этапах разработки:

Для освоения возможностей ESP-32 в рамках данного проекта были реализованы следующие задачи:

- 1. Управление светодиодами:**
 - Создана программа для мигания светодиодов и управления их яркостью с использованием PWM (широтно-импульсной модуляции). Это позволило изучить базовую работу с GPIO и принципами настройки ШИМ-сигналов.
- 2. Работа с датчиком расстояния:**
 - Проведён эксперимент с подключением ультразвукового датчика расстояния (например, HC-SR04). Код для ESP-32 обрабатывал данные от датчика и выводил результаты через последовательный порт. Это дало возможность изучить работу с интерфейсами I2C и обработкой аналоговых сигналов.

Программная среда PlatformIO для работы с ESP-32:

В процессе работы использовалась интеграционная среда PlatformIO, установленная в Visual Studio Code. Данное расширение существенно упрощает разработку проектов под микроконтроллеры, включая ESP-32.

Преимущества PlatformIO:

- 1. Автоматическое управление библиотеками:**
 - Простое добавление, обновление и удаление библиотек через встроенный менеджер.

2. Мультиплатформенная поддержка:

- Возможность работы с различными микроконтроллерами (Arduino, ESP32, STM32 и др.).

3. Интеграция отладки:

- Наличие инструментов для мониторинга последовательного порта и тестирования приложений.

4. Эффективное управление проектами:

- Простое переключение между различными конфигурациями и поддержка систем контроля версий, таких как Git.

ESP-32 является мощным и доступным инструментом для создания систем сбора, обработки и передачи данных. Благодаря высокой производительности, богатому набору интерфейсов и поддержке беспроводной связи, данная плата обеспечивает значительное преимущество при создании современных электронных устройств. Использование ESP-32 позволяет сосредоточиться на решении прикладных задач, минимизируя время и ресурсы, затрачиваемые на аппаратную реализацию.

2. Модуль AD8232

AD8232 — это специализированный интегральный модуль, предназначенный для обработки биопотенциалов, таких как электрокардиографические (ЭКГ) или электроэнцефалографические (ЭЭГ) сигналы. Благодаря своей компактности и простоте интеграции, AD8232(Рис. 2) широко применяется в медицинских и исследовательских проектах для мониторинга физиологических показателей.

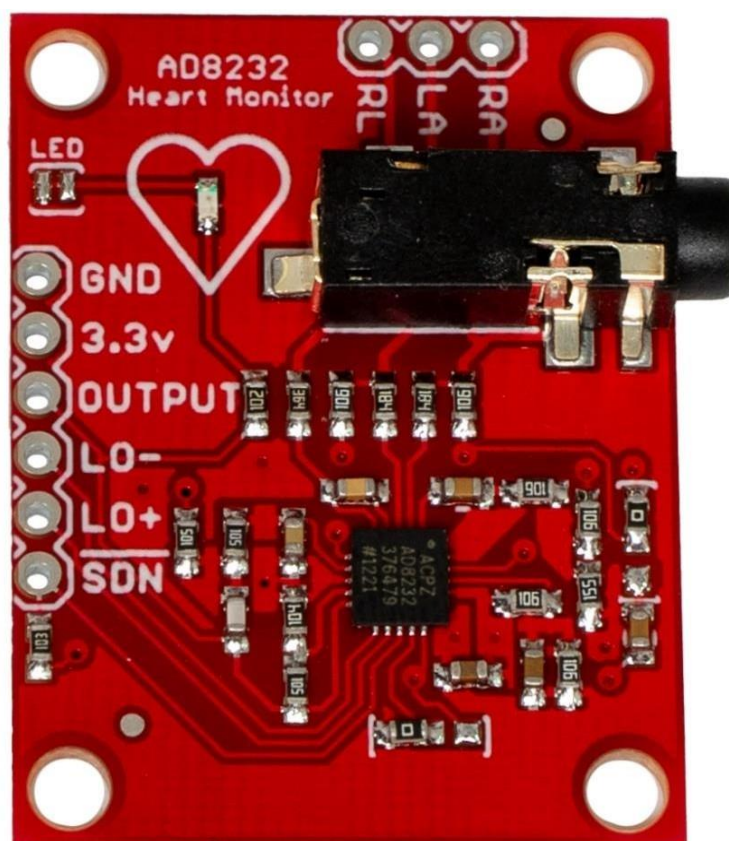


Рис. 2. Датчик сердечного ритма AD8232

Основные характеристики AD8232:

2.1. Функциональность:

2.1.1. Устройство представляет собой усилитель для биопотенциалов, оптимизированный для работы с малошумными сигналами.

2.2. Схема передней панели:

2.2.1. Встроенная фильтрация и усиление сигналов позволяют минимизировать помехи и шумы, связанные с движениями или электромагнитными источниками.

2.3. Низкое энергопотребление:

2.3.1. Идеально подходит для портативных и автономных систем.

2.4. Поддержка различных режимов:

2.4.1. Модуль может быть настроен для мониторинга в реальном времени или записи данных для последующего анализа.

Структура модуля AD8232:

Модуль включает:

- **AD8232 IC** — интегральная микросхема, выполняющая функции усиления и фильтрации.
- **Контактные площадки** для подключения датчиков (электродов).
- **Выводы** для подключения к микроконтроллеру или другому устройству (обычно используются выводы OUT, LO+, LO-, 3.3V и GND).
- **Индикаторы активности:** светодиоды для отображения статуса работы.

Преимущества AD8232:

1. Высокая чувствительность:

- Позволяет эффективно улавливать слабые электрические сигналы от человеческого тела.

2. Фильтрация шумов:

- Встроенные фильтры нижних частот снижают воздействие сетевых помех (например, 50/60 Гц).

3. Универсальность:

- Подходит для мониторинга различных типов биосигналов, таких как ЭКГ, ЭМГ и ЭЭГ.

4. Совместимость:

- Легко интегрируется с популярными микроконтроллерами, такими как Arduino или ESP-32.

Применение AD8232 в разработке комплекса:

1. Сбор данных ЭЭГ:

- Используется для регистрации слабых электрических сигналов мозга через электроды, закрепленные на коже головы.

2. Обработка сигналов в реальном времени:

- Усиление и фильтрация сигналов на этапе аппаратной обработки снижают нагрузку на микроконтроллер.

3. Интеграция с ESP-32:

- Полученные данные передаются на ESP-32 для последующей обработки, анализа или передачи на внешний интерфейс.

Подключение и работа с AD8232:

Для работы с модулем используются три электрода, которые размещаются на коже человека:

- **RA** (правая рука или область рядом с правой стороной головы).
- **LA** (левая рука или область рядом с левой стороной головы).
- **RL** (референтный электрод).

Сигналы обрабатываются внутри модуля и передаются через выходной контакт (OUT) для дальнейшей обработки.

Модуль AD8232 является ключевым компонентом в разработке систем сбора и анализа биопотенциалов. Его возможности по усилению и фильтрации сигналов, а также низкое энергопотребление делают его оптимальным решением для мобильных устройств и систем мониторинга здоровья. Использование AD8232 в сочетании с ESP-32 позволяет создавать функциональные и высокоточные комплексы для анализа ЭЭГ.

3. Программно-аппаратный комплекс

Для реализации программной части системы сбора данных с ЭЭГ-датчиков был разработан интерфейс взаимодействия с устройством на языке программирования C#. Выбор C# был обусловлен необходимостью создания удобного и гибкого интерфейса, а также наличием встроенных инструментов для работы с последовательными портами, что значительно упрощает реализацию связи с оборудованием.

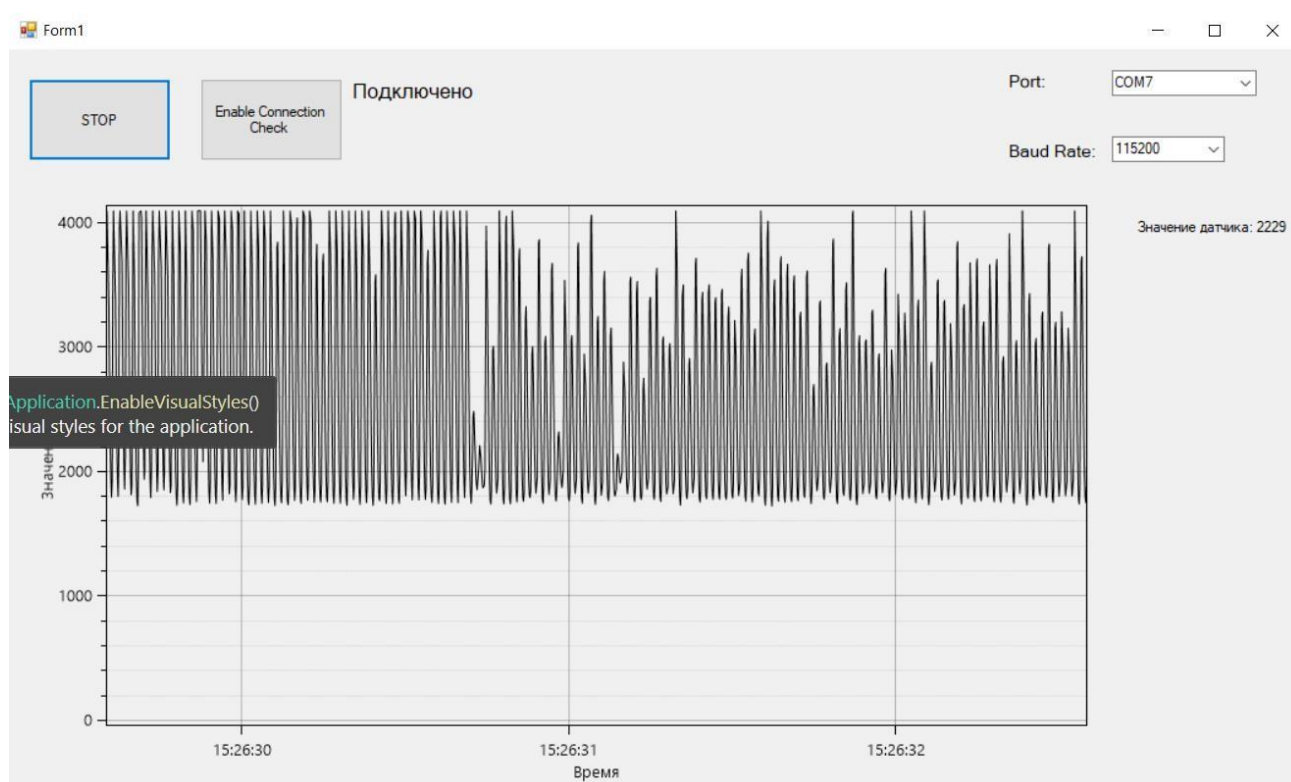


Рис. 3. Программный интерфейс

Реализация обработки данных с датчика

Программный модуль выполняет следующие основные функции:

- **Сбор данных:** Программа подключается к датчику через последовательный порт, считывает передаваемые данные и преобразует их в удобный формат для анализа.
- **Фильтрация данных:** Одной из ключевых задач при работе с данными ЭЭГ является минимизация шумов. Шумы могут возникать из-за различных факторов, включая электромагнитные помехи. Особенно часто встречается шум частотой 50 Гц, вызванный сетью переменного тока. Для его подавления в программный модуль была добавлена функция фильтрации.

Фильтрация сетевого шума частотой 50 Гц

Шум в частотном диапазоне 50 Гц может существенно исказить полученные данные ЭЭГ. Чтобы минимизировать это воздействие, в программе реализован алгоритм фильтрации данных. Для устранения сетевого шума может быть использован специальный **цифровой полосовой фильтр**, например, фильтр БИХ (бесконечный импульсный фильтр) или ФИР (конечный импульсный фильтр), настроенный на подавление частоты 50 Гц. В данном случае фильтрация позволяет оставить только те частоты, которые несут полезную информацию о мозговой активности, что критически важно для анализа.

Фильтрация проводится следующим образом:

- 3.1. Входной сигнал, получаемый через последовательный порт, подвергается первичной обработке для выделения данных в числовом формате.
- 3.2. Применяется алгоритм подавления помех на основе фильтров или адаптивных методов сглаживания.
- 3.3. Полученные данные обрабатываются и передаются для дальнейшего анализа или визуализации.

Реализация интерфейса связи и фильтрации данных на языке C# позволила обеспечить эффективное взаимодействие с ЭЭГ-датчиком. Программный модуль не только получает и обрабатывает данные, но и подавляет электромагнитные помехи, что значительно улучшает качество сигналов для последующего анализа. Такой подход делает систему надежной и пригодной для использования в исследовательских или прикладных целях.

4. Neurosky Mobile

В рамках разработки комплекса для получения электроэнцефалограммы (ЭЭГ) был проведен эксперимент с использованием устройства Neurosky Mobile. Этот прибор представляет собой компактный и удобный в использовании гаджет для регистрации электрической активности мозга, который широко применяется в образовательных и исследовательских целях.

Устройство оснащено встроенным сенсором, размещаемым на лбу пользователя, и передаёт данные о мозговой активности по Bluetooth. Оно способно регистрировать сигналы ЭЭГ в режиме реального времени и предоставляет основные параметры, такие как уровень внимания, медитации и другие метрики, полученные на основе обработки мозговых волн.

Дополнительно гарнитура использует встроенный чип ThinkGear, который выполняет предварительную обработку сигналов непосредственно на борту устройства. Это позволяет выделять характеристики волн различных частотных диапазонов (дельта, тета, альфа, бета и гамма), а также рассчитывать производные метрики — например, индекс концентрации или уровень расслабленности. Благодаря этому пользователю предоставляется упрощённый, но информативный набор данных, пригодный для приложений в сфере нейроинтерфейсов, обучения, медитации и взаимодействия с внешними устройствами.

Такая архитектура делает Neurosky Mobile привлекательным решением для задач, не требующих высокой точности медицинской диагностики, но позволяющим получить общее представление о нейрофизиологическом состоянии пользователя. Несмотря на упрощённую схему съёма сигнала (один электрод и референс), устройство демонстрирует стабильную работу в бытовых условиях, а его совместимость с мобильными платформами и компьютерами обеспечивает широкие возможности по визуализации, хранению и анализу данных. Благодаря открытым SDK и активному сообществу разработчиков, гарнитура легко интегрируется в пользовательские приложения, включая системы биоуправления, нейроигры, тренажёры внимания и образовательные платформы, ориентированные на изучение основ нейронаук.



Рис. 4. Нейро-гарнитура MindWave Mobile

Проведенные эксперименты

Для тестирования устройства был использован как официальный программный пакет, предоставляемый производителем, так и сторонний программный софт, опубликованный на GitHub. Официальное приложение позволило быстро настроить устройство, проверить основные параметры работы и получить базовые сигналы ЭЭГ. Интерфейс программы интуитивно понятен и включает визуализацию сигналов в реальном времени, а также возможность записи данных для последующего анализа.

Сторонний софт, найденный на GitHub, оказался полезным для более глубокого взаимодействия с устройством. Этот софт предоставил доступ к исходным данным ЭЭГ и позволил использовать нестандартные алгоритмы обработки сигналов. В частности, удалось настроить прямую передачу данных через Bluetooth и интегрировать устройство с пользовательскими программными решениями. Примеры приложений включали скрипты на Python для анализа сигналов и визуализации, а также подключение к системам машинного обучения для классификации данных ЭЭГ.

Возможности интеграции с ESP-32

Для дальнейшего использования устройства Neurosky Mobile в рамках

разрабатываемого комплекса планируется интеграция с микроконтроллером ESP-32. Этот модуль, оснащённый встроенным модулем Bluetooth, позволяет принимать данные от Neurosky Mobile и передавать их на другие устройства или в облачные сервисы для обработки.

Потенциальные направления интеграции:

1. **Реализация автономной системы сбора данных:** использование ESP-32 для сохранения данных ЭЭГ на внешнюю память или их предварительной обработки на борту микроконтроллера.
2. **Обработка и визуализация данных:** применение ESP-32 для передачи данных в реальном времени на мобильное приложение или ПК с использованием Wi-Fi.
3. **Управление устройствами на основе сигналов ЭЭГ:** создание интерфейсов мозг-компьютер (Brain-Computer Interfaces, BCI) для управления роботами, освещением или другими объектами на основе определённых паттернов активности мозга.

Таким образом, использование Neurosky Mobile совместно с ESP-32 открывает широкие возможности для создания компактных, мобильных и доступных систем сбора и анализа данных ЭЭГ. Это решение может быть востребовано в образовательной, исследовательской и прикладной сферах, включая медицину и развлекательные технологии.

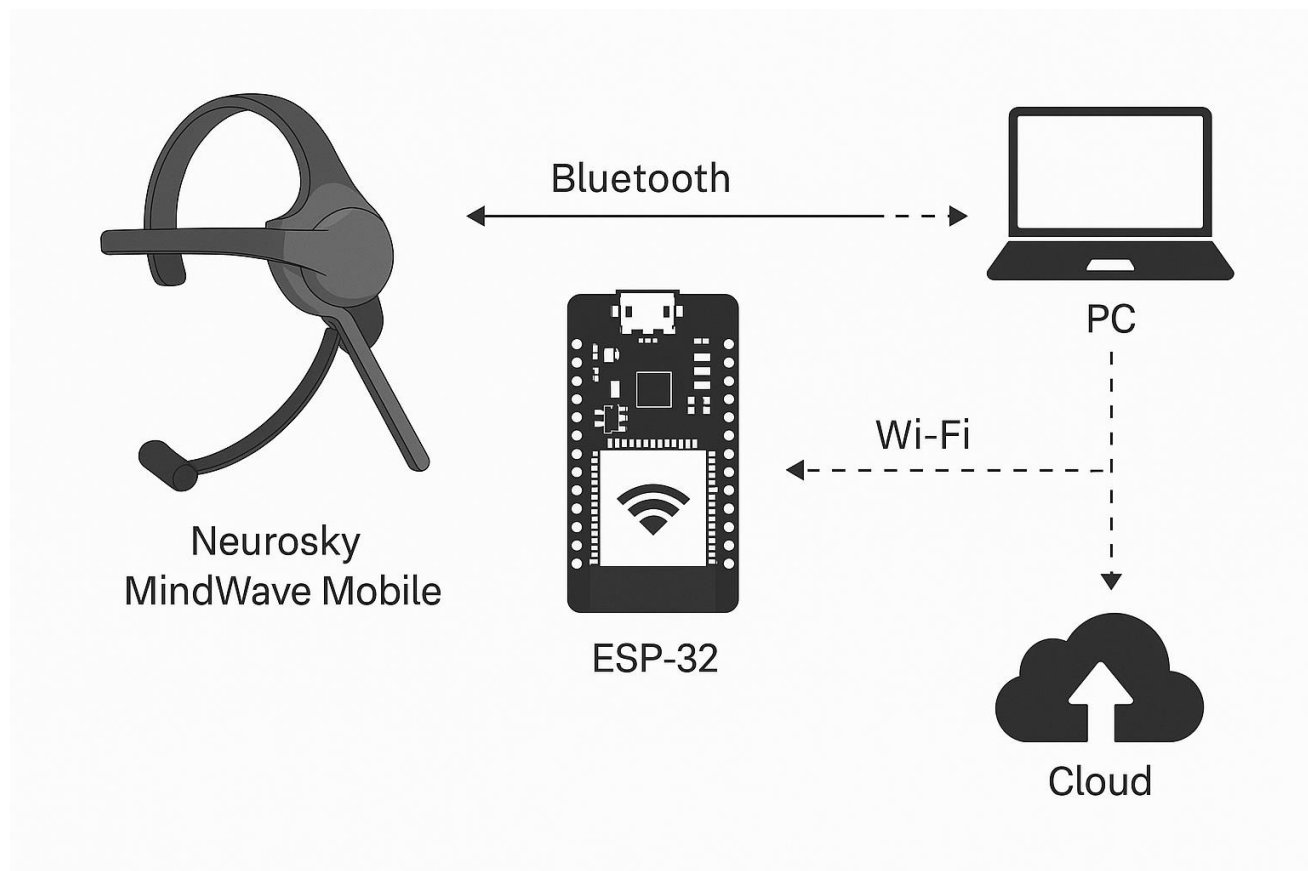


Рис. 5. Схема интеграции нейро-гарнитуры MindWave Mobile с микроконтроллером ESP-32

4.1 Мозговые волны и их значение в электроэнцефалографии

Мозговые волны представляют собой ритмические колебания электрической активности головного мозга, которые возникают в результате взаимодействия миллионов нейронов. Эти колебания регистрируются с помощью электроэнцефалографии (ЭЭГ) и служат важнейшим биомаркером различных психофизиологических состояний человека. Благодаря ЭЭГ исследователи могут наблюдать, как изменяется активность мозга в зависимости от состояния бодрствования, сна, концентрации внимания, эмоционального фона и даже медитативной практики.

Сигналы ЭЭГ можно условно разделить на несколько основных диапазонов частот, каждый из которых ассоциируется с определёнными ментальными и физиологическими состояниями. Эти диапазоны получили названия: **дельта**, **тета**, **альфа**, **бета**, **гамма**, а также **му-волны**, которые относятся к отдельной категории и чаще ассоциируются с моторной активностью и сенсомоторной интеграцией.

- **Delta-волны** (менее 4 Гц) — это самые медленные из всех мозговых волн. Они преобладают во время глубокого сна, особенно в фазе медленного сна (slow-wave sleep). Также такие волны наблюдаются у опытных медитаторов в состоянии глубокой расслабленности. Появление дельта-активности у бодрствующего человека может указывать на определённые нарушения сознания или повреждение мозговых структур.
- **Theta-волны** (4–7 Гц) часто связывают с состояниями лёгкой дремоты, медитации и полусна. Они характерны для фазы неглубокого сна и также могут наблюдаться при гипнотических состояниях. В некоторых случаях выраженная тета-активность у бодрствующего человека сопровождается ослаблением восприятия телесных ощущений и внешнего мира.
- **Alpha-волны** (7–13 Гц) являются индикатором спокойного бодрствования и расслабленности. Они появляются при закрытых глазах, в состоянии покоя, особенно когда человек не сосредоточен на каком-либо задании. Альфа-волны считаются «волнами медитации», поскольку активно проявляются в моменты внутреннего сосредоточения и умиротворения.
- **Mu-волны** (8–12 Гц) накладываются на диапазон альфа-волн, но регистрируются в области центральных извилин и имеют другое функциональное значение — они связаны с сенсомоторной активностью. Му-ритм подавляется, когда человек совершает движение или даже просто представляет себе моторную активность.
- **Beta-волны** (13–39 Гц) соответствуют активному, бодрствующему состоянию. Они характерны для процессов концентрации внимания, решения задач, обработки информации. Бета-ритм также может усиливаться при тревожности или внутреннем возбуждении.
- **Gamma-волны** (более 40 Гц) — самые высокочастотные из регистрируемых мозговых волн. Они ассоциируются с высокоуровневыми когнитивными процессами: осознанием, обработкой сложной информации, синхронизацией различных отделов мозга. Некоторые исследования указывают, что гамма-активность может возрастать во время духовных или медитативных практик.

Frequency range (Hz)	Name	Related attributes and states
>40	Gamma waves	Higher mental activity, perception, problem solving, fear, and consciousness. Appears in specific meditative states, relating to Buddhist compassion meditations in the Tibetan tradition
13–39	Beta waves	The most usual state for normal everyday consciousness. Active, busy or even anxious thinking. Also appears in active concentration, arousal, cognition, and or paranoia
7–13	Alpha waves	Relaxed wakefulness, pre-sleep and pre-wake drowsiness, REM sleep, dreams and creative thought or free association. Considered as the brainwaves of meditation. These waves also appear in the relaxation process before sleep
8–12	Mu waves	Sensorimotor rhythm, Mu rhythm
4–7	Theta waves	Appears in deep meditation/relaxation, NREM sleep. Also, in hypnotic states or where some element of consciousness. A theta prominent individual may be awake but lose their sense of bodily location, for example
<4	Delta waves	Deep dreamless sleep with loss of body awareness. Does appear in the EEG of very experienced practitioners of meditation and would appear to relate to some ecstatic states. Maintaining consciousness while delta present is difficult

Рис. 6. Частотная классификация мозговых волн и соответствующие психофизиологические состояния.

Таким образом, анализ частотных характеристик мозговой активности предоставляет ценную информацию о текущем состоянии центральной нервной системы. Именно эти параметры, извлекаемые из ЭЭГ, становятся основой для различных применений: от клинической диагностики до интерфейсов мозг-компьютер (BCI), а также в обучающих и развлекательных технологиях, использующих нейрообратную связь.

4.2 Протокол передачи данных устройства NeuroSky MindWave Mobile

Устройство NeuroSky MindWave Mobile использует специализированный протокол для передачи данных, основанный на потоковой передаче байтов по Bluetooth. Этот протокол позволяет взаимодействовать с внешними программами, собирая информацию о состоянии пользователя и электрической активности его мозга. В отличие от других нейроинтерфейсов, MindWave Mobile ориентирован на обеспечение простой интеграции с приложениями для нейрообратной связи (neurofeedback), мониторинга психофизиологических состояний, а также использования в образовательных и развлекательных целях.

Протокол передачи данных включает различные типы пакетов, каждый из которых имеет уникальный идентификатор (в шестнадцатеричном формате), что позволяет точно определить тип передаваемой информации. В таблице ниже приводится перечень кодов, соответствующих различным типам данных, передаваемых устройством:

Code	Byte length	Data value meaning
0x02	1	POOR_SIGNAL quality (0–255)
0x03	1	HEART_RATE (0–255)
0x04	1	ATTENTION eSense (0–100)
0x05	1	MEDITATION eSense (0–100)
0x06	1	8BIT_RAW wave value (0–255)
0x07	1	RAW_MARKER section start
0x80	2	RAW wave value
0x81	32	EEG_POWER: eight big-endian 4 byte floating point values representing the bands (delta, theta, low-alpha etc.)
0x83	24	ASIC_EEG_POWER: eight big endian 3-byte unsigned integer values representing delta, theta etc.
0x86	2	RRINTERVAL: two byte big endian unsigned integer representing the milliseconds between two R-peaks
0x55	–	Not used (reserved for [EXCODE])
0xAA	–	Not used (reserved for [SYNC])

Рис. 7. Пример программной реализации

Протокол предоставляет множество полезных данных, которые могут быть использованы для различных целей, включая медитацию, тренировки внимания, мониторинг психофизиологических состояний, а также для исследований и разработки нейротехнологий. Рассмотрим подробнее некоторые из этих данных.

1. POOR_SIGNAL (0x02)

Этот параметр отображает качество сигнала, поступающего с электродов устройства. Значение варьируется от 0 до 255, где 0 обозначает идеальный контакт с кожей, а значения выше 200 говорят о плохом контакте или помехах. Этот параметр имеет важное значение для корректной работы устройства, так как сигнал с низким качеством может привести к ошибкам в интерпретации данных.

2. ATTENTION (0x04) и MEDITATION (0x05)

Эти параметры дают информацию о психоэмоциональном состоянии пользователя, измеряя уровень концентрации и медитации. Значения варьируются от 0 до 100, где 0 соответствует

низкому уровню внимания или медитации, а 100 — высокому уровню. Эти показатели часто используются в приложениях для нейрообратной связи, где цель — улучшение навыков концентрации или достижения состояния расслабления.

3. EEG_POWER (0x81) и ASIC_EEG_POWER (0x83)

Пакеты с кодами 0x81 и 0x83 содержат данные о мозговой активности пользователя в разных диапазонах частот, от дельта до гамма-волны. Эти данные могут быть использованы для анализа состояния мозга и выявления паттернов, таких как уровни стресса или расслабления. Пакет 0x81 передает восемь 4-байтных значений с плавающей точкой, представляющих различные диапазоны мозговых волн. Альтернативный пакет 0x83 содержит аналогичную информацию, но в более компактной форме — с использованием целочисленных значений (по 3 байта на каждое значение).

4. 8BIT_RAW (0x06) и 16BIT_RAW (0x80)

Эти пакеты содержат необработанные данные ЭЭГ в 8-битном (0x06) или 16-битном (0x80) формате. Эти данные могут быть использованы для более глубокого анализа сигнала, например, с использованием методов машинного обучения или цифровой фильтрации для извлечения различных характеристик мозговой активности. Они особенно полезны в научных и исследовательских приложениях, где требуется точный контроль и обработка сырых данных.

5. HEART_RATE (0x03) и RRINTERVAL (0x86)

Эти параметры измеряют частоту сердечных сокращений и интервалы между ними. Параметр HEART_RATE дает информацию о частоте сердечных сокращений в диапазоне от 0 до 255, а RRINTERVAL — о времени между сокращениями в миллисекундах. Эти данные полезны для мониторинга здоровья пользователя, а также для исследования влияния физиологических состояний на мозговую активность.

Протокол передачи данных устройства NeuroSky MindWave Mobile предоставляет разработчикам гибкие возможности для получения разнообразных данных о состоянии мозга и физиологическом состоянии пользователя. Он включает как базовые метрики, такие как качество сигнала и уровень концентрации, так и более сложные данные о мозговой активности, которые могут быть использованы для научных исследований, разработки приложений для нейрообратной связи и медитации, а также для мониторинга состояния здоровья. Протокол дает разработчикам широкий спектр инструментов для создания инновационных решений в области нейротехнологий, психофизиологического мониторинга и улучшения пользовательского опыта.

4.3 Реализация и тестирование приёма данных от устройства MindWave

На данном этапе проекта был разработан программный модуль для приёма и обработки данных от нейроинтерфейса MindWave, подключённого к ПК через последовательный порт. Этот модуль представляет собой компонент, предназначенный для сбора данных о состоянии пользователя, таких как качество сигнала, уровень концентрации, частота сердечных сокращений и другие биометрические параметры, которые могут быть полезны для дальнейшего анализа.

Описание функциональности

Класс MindWaveReader реализует полный цикл взаимодействия с устройством:

1. Обнаружение COM-портов

При запуске программы она автоматически находит доступные COM-порты и предлагает пользователю выбрать один из них для подключения к устройству. Это позволяет пользователю гибко настроить взаимодействие с интерфейсом.

2. Установка соединения

После выбора порта программа устанавливает соединение с устройством MindWave, используя стандартную скорость 57600 бод, что является рекомендованной настройкой для этого устройства. Соединение осуществляется через последовательный порт, и далее происходит инициирование передачи данных.

3. Инициализация передачи данных

Для начала получения данных от устройства отправляется управляющий байт, который активирует поток передачи данных. Это позволяет начать считывание показаний с нейроинтерфейса.

4. Обработка входящего потока данных

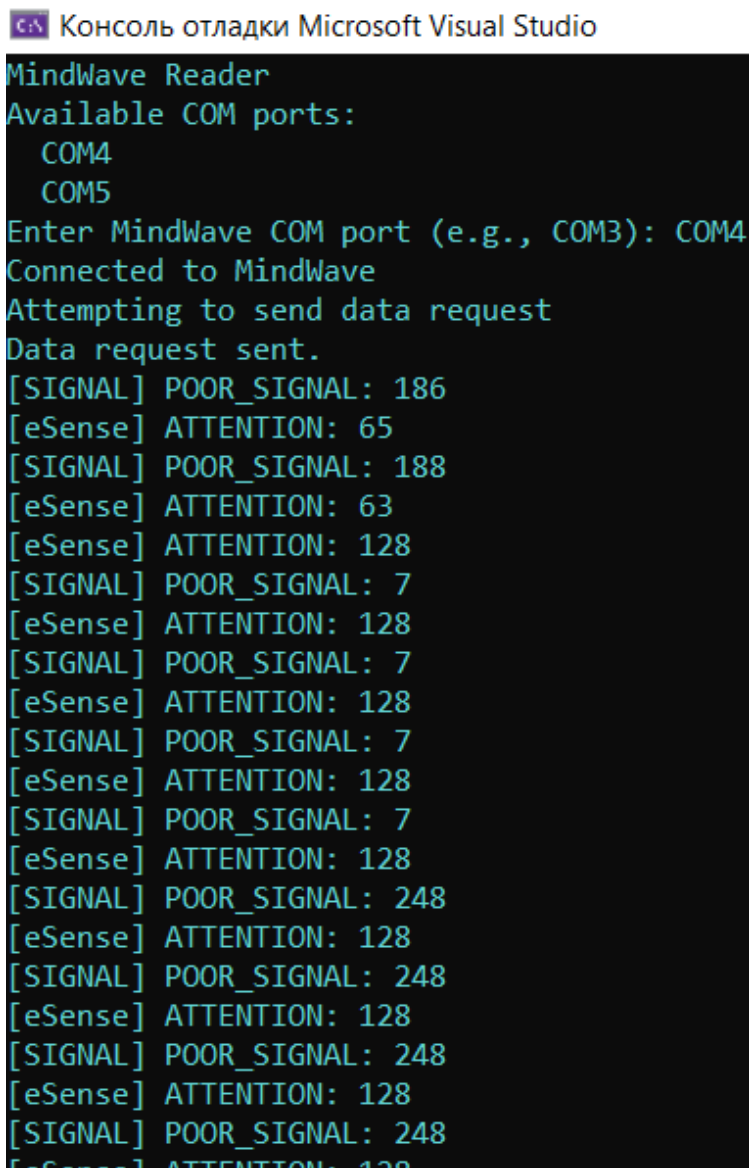
Программа в реальном времени распознаёт и декодирует пакеты данных, получаемые с устройства, в соответствии с протоколом ThinkGear. Это включает распознавание различных типов сигналов, таких как:

- POOR_SIGNAL — сигнал, отражающий качество связи с устройством (от 0 до 255, где 0 — идеальное качество).
- eSense: ATTENTION — уровень концентрации пользователя, который варьируется от 0 до 100.
- Другие типы сигналов, такие как медитация, ЭЭГ и пульс, также обрабатываются и могут быть добавлены для дальнейшей визуализации или сохранения.

5. Логирование данных

Все полученные данные записываются в текстовый файл. Имя файла формируется на основе текущей даты и времени, что позволяет сохранять данные в удобном формате для последующего анализа.

Демонстрация работы программы



```
cs Консоль отладки Microsoft Visual Studio
MindWave Reader
Available COM ports:
  COM4
  COM5
Enter MindWave COM port (e.g., COM3): COM4
Connected to MindWave
Attempting to send data request
Data request sent.
[SIGNAL] POOR_SIGNAL: 186
[eSense] ATTENTION: 65
[SIGNAL] POOR_SIGNAL: 188
[eSense] ATTENTION: 63
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 7
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 7
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 7
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 7
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 248
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 248
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 248
[eSense] ATTENTION: 128
[SIGNAL] POOR_SIGNAL: 248
[eSense] ATTENTION: 128
```

Рис. 8. Пример программной реализации

На рисунке 8 показан пример работы программы в режиме отладки в среде Microsoft Visual Studio. В консоли отображаются поступающие данные о качестве сигнала и уровне внимания в реальном времени, что демонстрирует успешное взаимодействие с устройством и получение актуальных данных.

Реализованный модуль успешно выполняет приём и интерпретацию сигнала с устройства MindWave. Он предоставляет возможность получать и обрабатывать биометрические данные в реальном времени, что может быть использовано как основа для дальнейших исследований или прикладных проектов в области нейроинтерфейсов. Например, эти данные могут быть использованы для создания более сложных систем визуализации, фильтрации или анализа, что открывает большие перспективы для дальнейшего развития системы.

Список литературы

1. C# 10 и .NET 6. Современная кроссплатформенная разработка. Автор: Марк Прайс. Издательство: Вильямс, 2022.
2. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. Автор: Джеффри Рихтер.
3. Kolban, N. Kolban's book on ESP32 / N. Kolban. – 1st ed. – [S.l.] : CreateSpace Independent Publishing Platform, 2017. – 560 p.
4. Gubler, M. Building Smart Drones with ESP8266 and Arduino / M. Gubler. – 1st ed. – [S.l.] : Apress, 2017. – 256 p. – ISBN 978-1-4842-3046-0.
5. Бутаков, В. В. Интернет вещей с ESP8266 / В. В. Бутаков. – М. : БХВ-Петербург, 2016. – 320 с. – ISBN 978-5-9775-0789-7.

Приложения

Приложение А. Реализация класса Device

```
public class Device
{
    private SerialPort _serialPort;
    public event Action<int> OnDataParsed;
    public event Action<string[]> PortsChanged;
    private string[] _ports = SerialPort.GetPortNames();

    public string[] Ports
    {
        get { return _ports; }
        set
        {
            if (!_ports.SequenceEqual(value))
            {
                _ports = value;
                PortsChanged?.Invoke(_ports);
            }
        }
    }

    public void CheckPorts()
    {
        string[] currentPorts = SerialPort.GetPortNames();
        Ports = currentPorts;
    }

    private bool _isConnected = false;

    public bool Connected
    {
        get { return _isConnected; }
        private set
        {
            if (value != _isConnected)
                _isConnected = value;
        }
    }

    private void HandleDataReceived(object sender,
    SerialDataReceivedEventArgs EventArgs)
    {
        string receivedData = _serialPort.ReadLine();
        Console.WriteLine($"Received Data: {receivedData}");

        if (int.TryParse(receivedData, out int parsedInteger))
        {
            OnDataParsed?.Invoke(parsedInteger);
        }
    }

    public void Connect(string portName, int baudRate)
    {
        if (_serialPort != null && _serialPort.IsOpen) _serialPort.Close();

        _serialPort = new SerialPort(portName) { BaudRate = baudRate };
        _serialPort.DataReceived += HandleDataReceived;
        try
```

```

        {
            _serialPort.Open();
            Connected = true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unexpected error: {ex.Message}");
            Connected = false;
        }
    }

    public void Disconnect()
    {
        if (_serialPort != null && _serialPort.IsOpen)
        {
            _serialPort.DataReceived -= HandleDataReceived;
            _serialPort.Close();
            Connected = false;
        }
    }
}

```

Приложение Б. Реализация класса Form

```

public partial class Form1 : Form
{
    private Device _device = new Device();
    private System.Windows.Forms.Timer _portCheckTimer;
    private NotchFilter filter;
    private bool _isReading = false;
    private string _lastSelectedPort = null;
    private PlotView _plotView;
    private LineSeries _lineSeries;

    public Form1()
    {
        InitializeComponent();
        InitializeChart();
        StartPosition = FormStartPosition.CenterScreen;
        string[] currentPorts = SerialPort.GetPortNames();
        UpdatePortList(currentPorts);
        LoadBaudRates();

        filter = new NotchFilter(50.0);
        _device.OnDataParsed += OnDeviceDataReceived;
        _device.PortsChanged += OnPortsChanged;

        _portCheckTimer = new System.Windows.Forms.Timer();
        _portCheckTimer.Interval = 1000;
        _portCheckTimer.Tick += (sender, e) => _device.CheckPorts();
        _portCheckTimer.Tick += (sender, e) => TryReconnect();
        _portCheckTimer.Start();

        port_comboBox.SelectedIndexChanged +=
port_comboBox_SelectedIndexChanged;
    }

    private void port_comboBox_SelectedIndexChanged(object sender, EventArgs
e)
    {

```

```

        if (!_device.Connected && _portCheckTimer.Enabled{
            OpenSelectedPort();
        }
    }

    private void OnPortsChanged(string[] ports)
    {
        Invoke(new Action(() => UpdatePortList(ports)));
    }

    private void TryReconnect()
    {
        if (_device.Connected && !string.IsNullOrEmpty(_lastSelectedPort))
        {
            _device.Connect(_lastSelectedPort,
(int)Baud_Rate_comboBox.SelectedItem);
            state_label.Text = "Автоматически переподключено";
            _isReading = true;
            start_button.Text = "STOP";
        }
    }

    private void UpdatePortList(string[] ports)
    {
        port_comboBox.Items.Clear();
        port_comboBox.Items.AddRange(ports);

        if (ports.Length > 0)
        {
            port_comboBox.SelectedIndex = 0;
        }
        else
        {
            state_label.Text = "Нет доступных портов";
            start_button.Text = "START";
        }
    }

    private void LoadBaudRates()
    {
        Baud_Rate_comboBox.Items.Clear();

        int[] baudRates = { 110, 300, 600, 1200, 2400, 4800, 9600, 19200,
38400, 57600, 115200, 230400, 460800 };

        Baud_Rate_comboBox.Items.AddRange(baudRates.Cast<object>().ToArray());
        Baud_Rate_comboBox.SelectedItem = 115200;
    }

    private void OpenSelectedPort()
    {
        if (port_comboBox.SelectedItem != null)
        {
            string selectedPort = port_comboBox.SelectedItem.ToString();
            _lastSelectedPort = selectedPort;
            try
            {
                _device.Connect(selectedPort,
(int)Baud_Rate_comboBox.SelectedItem);
                state_label.Text = "Подключено";
            }
            catch (Exception ex)

```



```

        {
            MessageBox.Show($"Не удалось открыть порт {selectedPort}:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

private void InitializeChart()
{
    _plotView = new PlotView
    {
        Location = new Point(10, 110),
        Size = new Size(900, 500)
    };

    var plotModel = new PlotModel();

    _lineSeries = new LineSeries
    {
        Color = OxyColors.Black,
        StrokeThickness = 1.5,
    };

    plotModel.Axes.Add(new DateTimeAxis
    {
        Position = AxisPosition.Bottom,
        Title = "Время",
        IntervalType = DateTimeIntervalType.Seconds,
        IntervalLength = 50,
        MajorGridlineStyle = LineStyle.Solid,
        MinorGridlineStyle = LineStyle.Dot,
    });

    plotModel.Axes.Add(new LinearAxis
    {
        Position = AxisPosition.Left,
        Title = "Значение",
        MajorGridlineStyle = LineStyle.Solid,
        MinorGridlineStyle = LineStyle.Dot,
    });

    plotModel.Series.Add(_lineSeries);

    _plotView.Model = plotModel;
    this.Controls.Add(_plotView);
}

private void ClearAndRedrawChart()
{
    _lineSeries.Points.Clear();
    _plotView.Model.InvalidatePlot(true);
}

private void OnDeviceDataReceived(int value)
{
    if (!_isReading) return;

    double filteredValue = filter.Apply(value);
    DateTime timeNow = DateTime.Now;

    value_label.BeginInvoke(new Action(() =>

```

```
{  
    Console.WriteLine(filteredValue);
```

```

        value_label.Text = $"Значение датчика: {value}";
        value_label.Update();
    }));

    _plotView.BeginInvoke(new Action(() =>
    {
        _lineSeries.Points.Add(new
DataPoint(DateTimeAxis.ToDouble(timeNow), value));

        var xAxis = _plotView.Model.Axes.FirstOrDefault(a => a is
DateTimeAxis);
        if (xAxis != null && _isReading)
        {
            xAxis.Minimum =
DateTimeAxis.ToDouble(DateTime.Now.AddSeconds(-3));
            xAxis.Maximum = DateTimeAxis.ToDouble(DateTime.Now);
        }

        _plotView.Model.InvalidatePlot(true);
    }));

private void start_button_Click(object sender, EventArgs e)
{
    if (!_isReading)
    {
        OpenSelectedPort();
        if (_device.Connected)
        {
            start_button.Text = "STOP";
            state_label.Text = "Подключено";
            _isReading = true;
        }
        else
        {
            state_label.Text = "Порт не найден";
        }
    }

    else if (_isReading && _device.Connected)
    {
        start_button.Text = "START";
        state_label.Text = "Остановлено";
        _isReading = false;

        ClearAndRedrawChart();
    }
}

private void button1_Click(object sender, EventArgs e)
{
    if (_portCheckTimer.Enabled)
    {
        _portCheckTimer.Stop();
        connection.Text = "Enable Connection Check";
    }
    else
    {
        _portCheckTimer.Start();
        connection.Text = "Disable Connection Check";
    }
}

```

}

Приложение В. Реализация класса Filter

```
public class NotchFilter
{
    private double[] a = new double[3];
    private double[] b = new double[3];

    private double[] inputPrev = new double[2];
    private double[] outputPrev = new double[2];

    public NotchFilter(double notchFreq)
    {
        double angularFreq = 2 * Math.PI * notchFreq / 500.0;
        double freqWidth = Math.Sin(angularFreq) / (2 * notchFreq);

        b[0] = 1;
        b[1] = -2 * Math.Cos(angularFreq);
        b[2] = 1;

        a[0] = 1 + freqWidth;
        a[1] = -2 * Math.Cos(angularFreq);
        a[2] = 1 - freqWidth;

        for (int i = 0; i < 3; i++)
        {
            b[i] /= a[0];
            a[i] /= a[0];
        }
    }

    public double Apply(double input)
    {
        double output = b[0] * input + b[1] * inputPrev[0] + b[2] * inputPrev[1]
            - a[1] * outputPrev[0] - a[2] * outputPrev[1];

        inputPrev[1] = inputPrev[0];
        inputPrev[0] = input;
        outputPrev[1] = outputPrev[0];
        outputPrev[0] = output;

        return output;
    }
}

public class SignalGenerator
{
    public static double[] GenerateSinSignal(double frequency, double
discretization, int count)
    {
        double[] signal = new double[count];

        for (int i = 0; i < count; i++)
        {
            signal[i] = Math.Sin(2 * Math.PI * frequency * i / discretization);
        }

        return signal;
    }
}
```

Приложение Г. Реализация класса MindWaveReader

```
class MindWaveReader
{
    static SerialPort serialPort;
    static StreamWriter txtWriter;
    static string txtFilePath;

    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Console.WriteLine("MindWave Reader");

        string exePath = AppDomain.CurrentDomain.BaseDirectory;
        string projectRoot = Path.GetFullPath(Path.Combine(exePath,
@"..\..\..\..\..\"));
        string dataFolder = Path.Combine(projectRoot, "data");

        txtFilePath = Path.Combine(dataFolder,
$"mindwave_data_{DateTime.Now:yyyyMMdd_HH:mm:ss}.txt");

        try
        {
            txtWriter = new StreamWriter(txtFilePath, false, Encoding.UTF8);
            Console.WriteLine($"Data will be saved to: {txtFilePath}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error creating file: {ex.Message}");
            return;
        }

        Console.WriteLine("Available COM ports:");
        string[] ports = SerialPort.GetPortNames();
        foreach (string port in ports)
        {
            Console.WriteLine("  " + port);
        }

        Console.Write("Enter MindWave COM port (e.g., COM3): ");
        string portName = Console.ReadLine();

        serialPort = new SerialPort(portName, 57600);
        serialPort.DataReceived += SerialPort_DataReceived;

        try
        {
            serialPort.Open();
            Console.WriteLine("Connected to MindWave");
            RequestData();
        }
        catch (Exception ex)
        {

```

```

        Console.WriteLine("Connection error: " + ex.Message);
        txtWriter?.Close();
        return;
    }

    Console.WriteLine("Press Enter to stop recording...");
    Console.ReadLine();

    serialPort.Close();
    txtWriter.Close();
    Console.WriteLine("Connection closed. Data saved to: " +
txtFilePath);
}

static void RequestData()
{
    Console.WriteLine("Attempting to send data request");
    byte[] request = new byte[] { 0xAA };
    serialPort.Write(request, 0, request.Length);
    Console.WriteLine("Data request sent.");
}

static void SerialPort_DataReceived(object sender,
SerialDataReceivedEventArgs e)
{
    var sp = (SerialPort)sender;
    int bytesToRead = sp.BytesToRead;
    byte[] buffer = new byte[bytesToRead];
    sp.Read(buffer, 0, bytesToRead);

    ParseThinkGearStream(buffer);
}

static void ParseThinkGearStream(byte[] data)
{
    for (int i = 0; i < data.Length; i++)
    {
        byte code = data[i];
        string valueStr = "";

        switch (code)
        {
            case 0x02: // POOR_SIGNAL quality (0-255)
                if (i + 1 < data.Length)
                {
                    valueStr = data[i + 1].ToString();
                    Console.WriteLine($"[SIGNAL] 0x{code:X2}:
{valueStr}");
                    WriteToTxt(code, valueStr);
                    i++;
                }
                break;

            case 0x03: // HEART_RATE (0-255)

```

```

        if (i + 1 < data.Length)
        {
            valueStr = data[i + 1].ToString();
            Console.WriteLine($"[HR] 0x{code:X2}: {valueStr}");
            WriteToTxt(code, valueStr);
            i++;
        }
        break;

    case 0x04: // ATTENTION eSense (0-100)
        if (i + 1 < data.Length)
        {
            valueStr = data[i + 1].ToString();
            Console.WriteLine($"[eSense] 0x{code:X2}:
{valueStr}");

            WriteToTxt(code, valueStr);
            i++;
        }
        break;

    case 0x05: // MEDITATION eSense (0-100)
        if (i + 1 < data.Length)
        {
            valueStr = data[i + 1].ToString();
            Console.WriteLine($"[eSense] 0x{code:X2}:
{valueStr}");

            WriteToTxt(code, valueStr);
            i++;
        }
        break;

    case 0x06: // 8BIT_RAW wave value (0-255)
        if (i + 1 < data.Length)
        {
            valueStr = data[i + 1].ToString();
            Console.WriteLine($"[EEG] 0x{code:X2}: {valueStr}");
            WriteToTxt(code, valueStr);
            i++;
        }
        break;

    case 0x07: // RAW_MARKER section start
        valueStr = "1";
        Console.WriteLine($"[MARKER] 0x{code:X2}");
        WriteToTxt(code, valueStr);
        break;

    case 0x80: // RAW wave value (2 bytes)
        if (i + 2 < data.Length)
        {
            short rawValue = (short)((data[i + 1] << 8) | data[i
+ 2]);

            valueStr = rawValue.ToString();
            Console.WriteLine($"[EEG] 0x{code:X2}: {valueStr}");

```

```

        WriteToTxt(code, valueStr);
        i += 2;
    }
    break;

case 0x81: // EEG_POWER (8x4 bytes)
    if (i + 32 < data.Length)
    {
        Console.WriteLine("[EEG] EEG_POWER:");
        for (int j = 0; j < 8; j++)
        {
            byte[] floatBytes = new byte[4]
            {
                data[i + 1 + j * 4],
                data[i + 2 + j * 4],
                data[i + 3 + j * 4],
                data[i + 4 + j * 4]
            };

            if (BitConverter.IsLittleEndian)
            {
                Array.Reverse(floatBytes);
            }

            float value = BitConverter.ToSingle(floatBytes,
0);

            valueStr =
value.ToString(CultureInfo.InvariantCulture);
            Console.WriteLine($" 0x{code:X2}: {valueStr}");
            WriteToTxt(code, valueStr);
        }
        i += 32;
    }
    break;

case 0x83: // ASIC_EEG_POWER (8x3 bytes)
    if (i + 24 < data.Length)
    {
        Console.WriteLine("[EEG] ASIC_EEG_POWER:");
        for (int j = 0; j < 8; j++)
        {
            int offset = i + 1 + j * 3;
            uint value = (uint)((data[offset] << 16) |
(data[offset + 1] << 8) | data[offset + 2]);
            valueStr = value.ToString();
            Console.WriteLine($" 0x{code:X2}: {valueStr}");
            WriteToTxt(code, valueStr);
        }
        i += 24;
    }
    break;

case 0x86: // RRINTERVAL (2 bytes)
    if (i + 2 < data.Length)

```



```

        {
            ushort rrInterval = (ushort)((data[i + 1] << 8) |
data[i + 2]);

            valueStr = rrInterval.ToString();
            Console.WriteLine($"[HR] 0x{code:X2}: {valueStr}
ms");

            WriteToTxt(code, valueStr);
            i += 2;
        }
        break;

    case 0x55: // EXCODE (reserved)
        valueStr = "1";
        Console.WriteLine($"[SYSTEM] 0x{code:X2}");
        WriteToTxt(code, valueStr);
        break;

    case 0xAA: // SYNC (reserved)
        valueStr = "1";
        Console.WriteLine($"[SYSTEM] 0x{code:X2}");
        WriteToTxt(code, valueStr);
        break;

    default:
        valueStr = "1";
        Console.WriteLine($"[UNKNOWN] 0x{code:X2}");
        WriteToTxt(code, valueStr);
        break;
    }
}

static void WriteToTxt(byte code, string value)
{
    int decimalCode = code;
    string line = $"{decimalCode},{value}";
    txtWriter.WriteLine(line);
    txtWriter.Flush();
}
}

```