Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
| *К защите  допустить*: |
| Заведующая кафедрой информатики |
| \_\_\_\_\_\_\_\_\_\_\_\_Н. А. Волорова |

Пояснительная записка

к дипломному проекту

на тему:

**ПРОГРАММНОЕ Обеспечение  
НАСТРОЙКИ И ДИАГНОСТИКИ ОТДЕЛЬНЫХ УЗЛОВ ПРИЕМО-ПЕРЕДАЮЩЕГО МОДУЛЯ**

БГУИР ДП 1-40 04 01 046 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | И. Д. Кондрашов |
| Руководитель |  | И. И. Гламаздин |
| Консультанты: |  |  |
| *от кафедры информатики* |  | И. И. Гламаздин |
| *по экономической части* |  | Т. А. Рыковская |
|  |  |  |
| Нормоконтролер |  | Н.Ю. Гриценко |
|  |  |  |
| Рецензент |  |  |

Минск 2023

**РЕФЕРАТ**

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НАСТРОЙКИ И ДИАГНОСТИКИ ОТДЕЛЬНЫХ УЗЛОВ ПРИЕМО-ПЕРЕДАЮЩЕГО МОДУЛЯ : дипломный проект / И.Д.Кондрашов. – Минск: БГУИР, 2023, – п.з – 76 с., чертежей (плакатов) – 6 л.

Ключевые слова: НАСТОЛЬНОЕ ПРИЛОЖЕНИЕ, WPF, MVVM, COM-ПОРТ, CAN-ШИНА, C#, SERIAL.

Объектом разработки является настольное приложение для диагностики аппаратных устройств.

Целью проекта является настольное приложение для настройки и диагностики аппаратных устройств.

При разработке приложения использовался следующий набор технологий: C#, WPF, COM-ПОРТ, CAN, MVVM, DESKTOP.

В первом разделе данной работы осуществляется анализ существующих аналогов и формирование требований к разрабатываемому приложению.

Второй раздел содержит информацию о технологиях, используемых для разработки данного настольного приложения.

В третьем разделе приведен процесс проектирования архитектуры настольного приложения, протокола обмена данными.

Четвертый раздел содержит описание процесса разработки приложения, а именно пользовательского интерфейса, реализация протокола обмена данными.

Пятый раздел содержит описание процесса тестирования основной функциональности приложения.

В шестом разделе приведено технико-экономическое обоснование эффективности разработки программного средства.

В разделения заключение содержатся выводы по данному дипломному проекту.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития приложения и наращивание его функционала.

**АННОТАЦИЯ**

на дипломный проект «Программное обеспечение настройки и диагностики отдельных узлов приемо-передающего модуля» студента учреждения образования «Белорусский государственный университет информатики и радиоэлектроники» Кондрашова И. Д.

Темой дипломного проекта является программное обеспечение настройки и диагностики отдельных узлов приемо-передающего модуля.

Во введении определена актуальность темы дипломного проекта, описано для чего нужна разрабатываемое программное обеспечение и кем она может использоваться.

В первой главе дипломного проекта дается обзор предметной области, анализ требований для дальнейшей разработки настольного приложения, описаны существующие аналоги, определены цели дипломного проекта.

Во второй главе проводится разбор используемых технологий в разработке дипломного проекта.

В третьей главе приводится протокол обмена данными для реализации программного обеспечения.

Четвертая глава посвящена программной реализации протокола обмена данными и клиентской части приложения.

В пятой главе приводится тестирование программного средства по заранее установленным требованиям.

В шестой главе приводится технико-экономическое обоснование разработанного проекта.

В заключении подводятся итоги и делаются выводы по дипломному проекту.

Министерство образования Республики Беларусь

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Учреждение образования БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Факультет | КСиС | | | Кафедра | | | информатики | | | | | | | | | | |
| Специальность | 1-40 04 01 | | | Специализация | | | | | |  | | | | | | | |
| УТВЕРЖДАЮ | | | | | | | | | | | | | | | | | |
|  | | | | | | | | |  | | | | | Н.А.Волорова | | | |
| « | | | | | | | | |  | | » | |  | | | 20 | г. |
| **ЗАДАНИЕ** | | | | | | | | | | | | | | | | | |
| **по дипломному проекту студента** | | | | | | | | | | | | | | | | | |
| Кондрашова Ивана Денисовича | | | | | | | | | | | | | | | | | |
| (фамилия, имя, отчество) | | | | | | | | | | | | | | | | | |
| 1. Тема проекта: | | Программное обеспечение настройки и диагностики отдельных узлов | | | | | | | | | | | | | | | |
| приемо-передающего модуля | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| утверждена приказом по университету от | | | | « | 17 | » | | марта | | | | 2023 г. | | | № | 673-C | |
| 2. Срок сдачи студентом законченной работы | | | | | 1 июня 2023 года | | | | | | | | | | | | |
| 3. Исходные данные к проекту | | | Язык программирования C#, интерфейс COM(RS-232), | | | | | | | | | | | | | | |
| интерфейс CAN, система построения настольных приложений WPF, интегрированная среда разработки Visual Studio. | | | | | | | | | | | | | | | | | |
| Цель проекта: разработать программное обеспечение настройки и диагностики отдельных узлов приемо-передающего модуля. | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| 4. Содержание пояснительной записки (перечень подлежащих разработке вопросов) | | | | | | | | | | | | | | | | | |
| Введение | | | | | | | | | | | | | | | | | |
| 1 Анализ аналогов и формирование требований к программному средству | | | | | | | | | | | | | | | | | |
| 2 [Обзор](#_tyjcwt) используемых технологий | | | | | | | | | | | | | | | | | |
| 3 Системное проектирование | | | | | | | | | | | | | | | | | |
| 4 Программная реализация | | | | | | | | | | | | | | | | | |
| 5 Тестирование программного средства | | | | | | | | | | | | | | | | | |
| 6 Технико-экономическое обоснование разработки и использования приложения | | | | | | | | | | | | | | | | | |
| Заключение | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |

|  |  |  |
| --- | --- | --- |
| 5. Перечень графического материала | | |
| Формирование команд управления приемо-передающим модулем – Схема программы – формат А4, лист 1. | | |
| Разбор диагностического пакета данных – Схема программы – формат А4, лист 1. | | |
| Формирование пакета данных CAN шины – Схема программы – формат А4, лист 1. | | |
| Отображение данных – Плакат – формат А4, лист 1. | | |
| Взаимодействие модулей аппаратно-программного комплекса – Плакат – формат А4, лист 1. | | |
| Диаграмма вариантов использования – Плакат – формат А4, лист 1. | | |
|  | | |
| 6. Технико-экономическое обоснование | | |
| Краткая характеристика программного продукта | | |
| Расчет затрат на разработку и цена программного обеспечения | | |
| Оценка результата от разработки программного обеспечения | | |
| Расчет показателей эффективности инвестиций в разработку программного обеспечения | | |
| Задание выдал |  | / Т.А. Рыковская / |
|  | | |

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов дипломного проекта (работы) | Объём этапа в % | Срок выполнения этапа | Примечание |
| Получения задания на дипломный проект |  | 01.02 |  |
| Обзор предметной области | 10 | 22.03 – 19.04 |  |
| Анализ функциональных требований и архитектуры программного средства | 15 | 12.02 – 23.02 |  |
| Разработка программного средства | 25 | 23.02 – 26.03 |  |
| Тестирование и отладка | 10 | 26.03 – 02.04 |  |
| Расчет экономической эффективности | 15 | 01.04 – 20.04 |  |
| Оформление пояснительной записки и |  |  |  |
| графического материала | 15-20 | 20.04 – 31.05 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Дата выдачи задания | 01.02.2023 | |  | |  | /И.И. Гламаздин/ | |
| Задание принял к исполнению | |  | | / И.Д. Кондрашов/ | | |

**СОДЕРЖАНИЕ**

[Введение 9](#_Toc135936690)

[1 Анализ аналогов и формирование требований к программному средству 11](#_Toc135936691)

[1.1 Обзор существующих аналогов 11](#_Toc135936692)

[1.1.1 Titan CAN Test 11](#_Toc135936693)

[1.1.2 PCAN-View 14](#_Toc135936694)

[1.1.3 Device Monitoring Studio 17](#_Toc135936695)

[1.2 Вывод по анализу и формирование требований 20](#_Toc135936696)

[2 Обзор используемых технологий 21](#_Toc135936697)

[2.1 Концептуальные сведения 21](#_Toc135936698)

[2.1.1 Настольное приложение 21](#_Toc135936699)

[2.1.2 Монолитная архитектура 21](#_Toc135936700)

[2.1.3 MVVM 21](#_Toc135936701)

[2.1.4 Язык программирования C# 22](#_Toc135936702)

[2.1.5 WPF 23](#_Toc135936703)

[2.1.6 SQLite 24](#_Toc135936704)

[2.1.7 COM-порт 24](#_Toc135936705)

[2.1.8 CAN 26](#_Toc135936706)

[3 Системное проектирование 28](#_Toc135936707)

[3.1 Общие сведения 28](#_Toc135936708)

[3.2 Модуль взаимодействия с базой данных 28](#_Toc135936709)

[3.3 Модуль графического интерфейса 28](#_Toc135936710)

[3.4 Модуль формирования пакета данных CAN-шины 29](#_Toc135936711)

[4 Программная реализация 38](#_Toc135936712)

[4.1 Формирование пакета данных CAN-шины 38](#_Toc135936713)

[4.1.1 Конфигурация подключения по COM-порту 38](#_Toc135936714)

[4.1.2 Формирование пакета данных команды настройки устройства 38](#_Toc135936715)

[4.1.3 Формирование пакета данных команды запроса параметров 39](#_Toc135936716)

[4.2 Чтение и запись данных по COM-порту 39](#_Toc135936717)

[4.3 Формирование протокола передачи данных по COM-порту 40](#_Toc135936718)

[4.4 Модуль взаимодействия с базой данных 40](#_Toc135936719)

[4.5 Клиентская часть 41](#_Toc135936720)

[4.6 Внедрение зависимостей 42](#_Toc135936721)

[5 Тестирование программного средства 44](#_Toc135936722)

[6 Технико-экономическое обоснование разработки и реализации программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля 49](#_Toc135936723)

[6.1 Краткая характеристика программного продукта 49](#_Toc135936724)

[6.2 Расчет затрат на разработку и цена программного обеспечения 49](#_Toc135936725)

[6.2.1 Затраты на основную заработную плату разработчиков 49](#_Toc135936726)

[6.2.2 Затраты на дополнительную заработную плату разработчиков 50](#_Toc135936727)

[6.2.3 Отчисления на социальные нужды 51](#_Toc135936728)

[6.2.4 Прочие затраты 51](#_Toc135936729)

[6.2.5 Полная сумма затрат на разработку программного обеспечения 51](#_Toc135936730)

[6.2.6 Плановая прибыль 52](#_Toc135936731)

[6.2.7 Отпускная цена программного средства 52](#_Toc135936732)

[6.3 Оценка результата от разработки программного обеспечения 53](#_Toc135936733)

[6.4 Расчет показателей эффективности инвестиций в разработку программного обеспечения 53](#_Toc135936734)

[Заключение 55](#_Toc135936735)

[Список используемых источников 56](#_Toc135936736)

[Приложение А 57](#_Toc135936737)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ**

В данной пояснительной записки применяются следующие сокращения и определения:

ПО – программное обеспечение.

СУБД – система управления базой данных.

COM-порт – communications port – интерфейс стандарта RS-232.

ID – identifier – идентификатор.

MVVM – Model-View-ViewModel – шаблон проектирования архитектуры приложения.

LINQ – language integrated query – язык запросов к источнику данных.

SQL – structured query language – структурированный язык запросов.

XML – extensible markup language – расширяемый язык разметки для документов.

XAML – extensible Application Markup Language – язык разметки, используемый для инициализации объектов в технологиях на платформе .NET.

WPF – windows presentation foundation – система построения клиентских приложений Windows.

DI – dependency injection – внедрение зависимостей.

API – Application Programming Interface – интерфейс программирования приложений.

CAN – controller area network – сеть контроллеров.

ЦМР – цена младшего разряда.

# **ВВЕДЕНИЕ**

В современном мире уже невозможно представить устройства, которые бы не управлялись программным обеспечением. Ранее оборудование управлялась аппаратным обеспечением, но теперь, когда чаще всего функционируют большие комплексы оборудования, уже не обойтись без программного обеспечения, предоставляющего возможность контроля над устройством.

Приемо-передающий модуль является одним из видов информационных систем, поэтому, чтобы дать для него определение, требуется сказать, что такое информационная система.

Информационная система – это система, осуществляющая: получение входных данных; обработку этих данных и/или изменение собственного внутреннего состояния (внутренних связей/отношений); выдачу результата либо изменение своего внешнего состояния (внешних связей/отношений).

Приемо-передающий модуль является сложной информационной, т. е. системой, которая содержит элементы, функционирующие в соответствии с правилами, порожденными отличными друг от друга множествами аксиом. При этом допускается, что среди правил функционирования различных элементов могут быть противоречащие друг другу правила и цели.

С другой стороны, приемо-передающий модуль – это прибор, предназначенный для обработки, расшифровки, отправки сигналов высокой и низкой частоты. Эти модули используют в системах радио- и спутниковой связи, радиорелейных линиях, активных фазированных антенных решетках (АФАР), разных видах радаров и лидаров. В состав этого устройства входят усилитель, передающий и приемные каналы, расположенные отдельно и разделенные с помощью циркулятора и коммутатора. Изделие после сборки может быть сразу подключено к аппаратному комплексу, принимать и передавать СВЧ сигналы без какого-либо дополнительного оборудования, т. е. приемо-передающий модуль является полностью автономным устройством.

Программное обеспечение для приемо-передающего модуля позволяет калибровать такие его параметры как: зависимость выходной мощности от входной, выходной мощности от температуры окружающей среды, стабильность фазы выходного сигнала в разных температурных условиях.

Таким образом, совокупность приемо-передающего модуля и программного обеспечения позволяет создать настроенный и откалиброванный аппаратно-программный комплекс, который может быть востребован на рынке радарных устройств.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности удовлетворяет требованиям кафедры информатики. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

# **1 АНАЛИЗ АНАЛОГОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ**

## **1.1 Обзор существующих аналогов**

Для того чтобы провести анализ существующих ПО настройки и диагностики оборудования, подключаемого через COM-порт, определим несколько критериев, по которым будут оцениваться рассматриваемые ПО:

* дизайн;
* наполненность интерфейса;
* доступность использования;
* предоставляемые функции.

**1.1.1** Titan CAN Test

Специализированное ПО для обмена данными через COM-порт Titan CAN Test было создано тайваньской компанией Titan в сентябре 2019 года.

**Дизайн**

Дизайн Titan CAN Test не является прорывным или оригинальным, явно отличающимся от других, но в то же время соответствует современным требованиям пользователя соответствующего ПО (рисунок 1.1):

* компактен, но не нагромождён;
* все требуемые функциональные элементы располагаются «под рукой»;
* хорошо подобранные неконтрастирующие цвета.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.1 – дизайн ПО Titan CAN Test

**Наполненность интерфейса**

При запуске приложения пользователю открывается окно, разделенное на 5 блоков:

* блок настройки подключения к COM-порту (рисунок 1.2);



Рисунок 1.2 – блок подключения к COM-порту

* таблица полученных через COM-порт пакетов данных (рисунок 1.3);

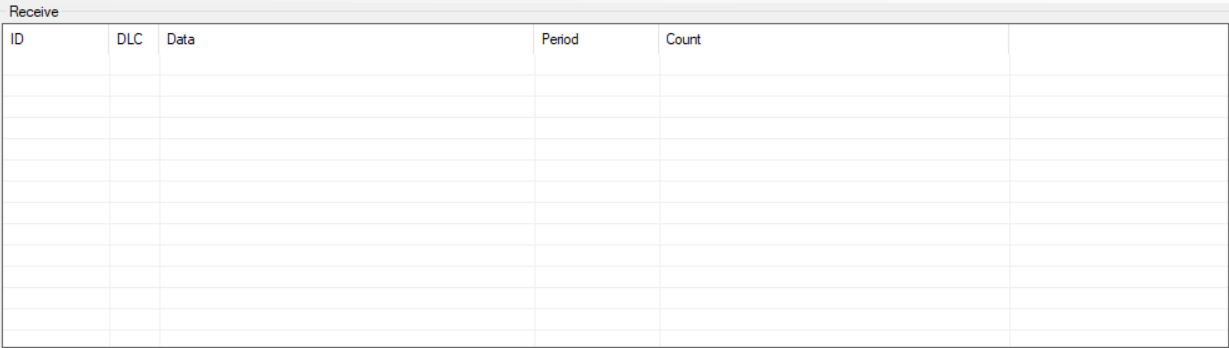


Рисунок 1.3 – таблица принятых пакетов данных

* таблица отправленных через COM-порт данных (рисунок 1.4);

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1.4 – таблица отправленных через COM-порт пакетов данных

* формирование вручную пакета данных и настройка его отправления (рисунок 1.5);

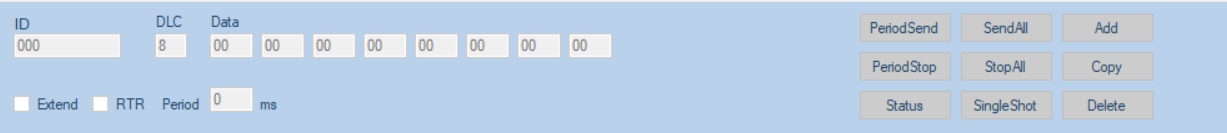
****

Рисунок 1.5 – блок формирования и отправки пакетов данных

* блок информирования о состоянии подключения (рисунок 1.6).

****

Рисунок 1.6 – блок информирования о состоянии подключения

**Доступность использования**

ПО Titan CAN Test можно скачать на официальном сайте производителя, что делает его бесплатным, хоть и работать оно будет только в паре с произведенной ими соответствующей серией адаптеров «USB-CAN».

**Предоставляемые функции Titan CAN Test**

* мгновенное подключение к устройству по COM-порту. Требуется выбрать в настройках требуемый COM-порт, скорость работы по выбранному порту, необязательные опции и ПО сразу выполнит подключение;
* сессии подключений. После выполнения определенной работы можно сохранить процесс, чтобы вернуться к определенному моменту после следующего запуска программы, а не начинать с начала;
* состояние подключения. После подключения к выбранному COM-порту в нижнем блоке приложения будет выводиться порт и скорость подключения, версия адаптера, выбран ли фильтр на отображение данных, режим работы адаптера, статус;
* разные варианты отправления данных. Сформировав пакет с данными, можно отправить их единожды, с заданным периодом времени, отправить несколько пакетов сразу, предварительно добавив их в таблицу отправляемых пакетов данных.

**Вывод**

На рынке узконаправленного ПО для настройки и диагностики оборудования через COM-порт Titan CAN Test является хорошим примером того, как должна выглядеть такая программа. Она проста и понятна, как и профессионалу, работающему с таким ПО каждый день, так и новичку, разбирающемуся в данной сфере, но не работавшему с таким ПО.

**1.1.2** PCAN-View

Узконаправленное ПО для обмена данными через COM-порт PCAN-View было создано немецкой компанией Peak System. Самая ранняя версия доступная на официальном сайте – version 3.0.0, датируемая 26 марта 2010 года.

**Дизайн**

Дизайн PCAN-View является более современным, чем дизайн ранее рассматриваемого аналога (рисунок 1.7):

* компактен и неочевиден, но любой пользователь от новичка до профессионала сможет быстро разобраться;
* на главный экран вынесено только самое необходимое;
* яркие и неконтрастирующие между собой цвета.

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Рисунок 1.7 – дизайн ПО PCAN-View

**Наполненность интерфейса**

При запуске приложения пользователю открывается окно, разделенное на 4 блока:

* блок подключения к COM-порту (рисунок 1.8);



Рисунок 1.8 – блок подключения к COM-порту

* таблица полученных через COM-порт пакетов данных (рисунок 1.9);

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.9 – таблица принятых пакетов данных

* таблица отправленных через COM-порт данных (рисунок 1.10);

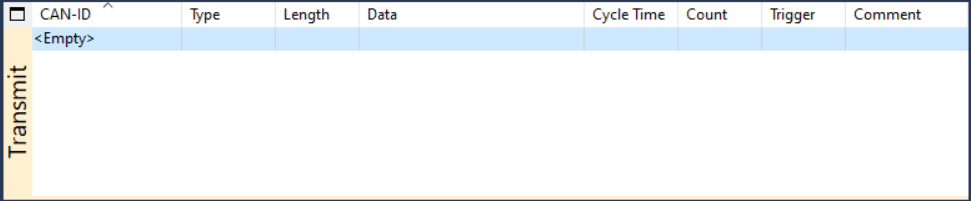


Рисунок 1.10 – таблица отправленных через COM-порт пакетов данных

* блок информирования о состоянии подключения (рисунок 1.11).



Рисунок 1.11 – блок информирования о состоянии подключения

**Доступность использования**

ПО PCAN-View можно скачать на официальном сайте производителя, что делает его бесплатным, хоть и работать оно будет только в паре с произведенными этой компанией устройствами.

**Предоставляемые функции PCAN-View**

* подключение к устройству по COM-порту. Требуется выбрать в настройках требуемый COM-порт, частоту обновления, формат ID(стандартный или расширенный) пакета данных, опциональные фильтры ID;
* сессии подключений. После выполнения определенной работы можно сохранить процесс, чтобы вернуться к определенному моменту после следующего запуска программы;
* состояние подключения. После подключения к выбранному COM-порту в нижнем блоке приложения будет отображаться подключено устройство или нет;
* формирование пакетов данных вручную. Во вкладке «New message» можно составить пакет данных задав ID, длину пакета данных, данные. Также можно выбрать период отправки и добавить комментарий к отправляемому пакету данных.

**Вывод**

На рынке специализированного ПО для настройки и диагностики оборудования через COM-порт PCAN-View является хорошим конкурентом, среди прочих ПО, с красивым дизайном и достаточным набором функций для работы специалиста.

**1.1.3** Device Monitoring Studio

Универсальное специализированное ПО для обмена данными и отслеживания работы устройств через COM-порт Device Monitoring Studio было выпущено на рынок компанией HHD Software уже более 9 лет назад.

**Дизайн**

Дизайн Device Monitoring Studio соответствует современным требования (рисунок 1.12):

* нагружен, весь функционал располагается на главном экране, но при этом имеет много свободного места;
* из-за загруженности функционалом неопытному пользователю будет трудно разобраться;
* не выделяющиеся цвета, которые не отвлекают при работе.

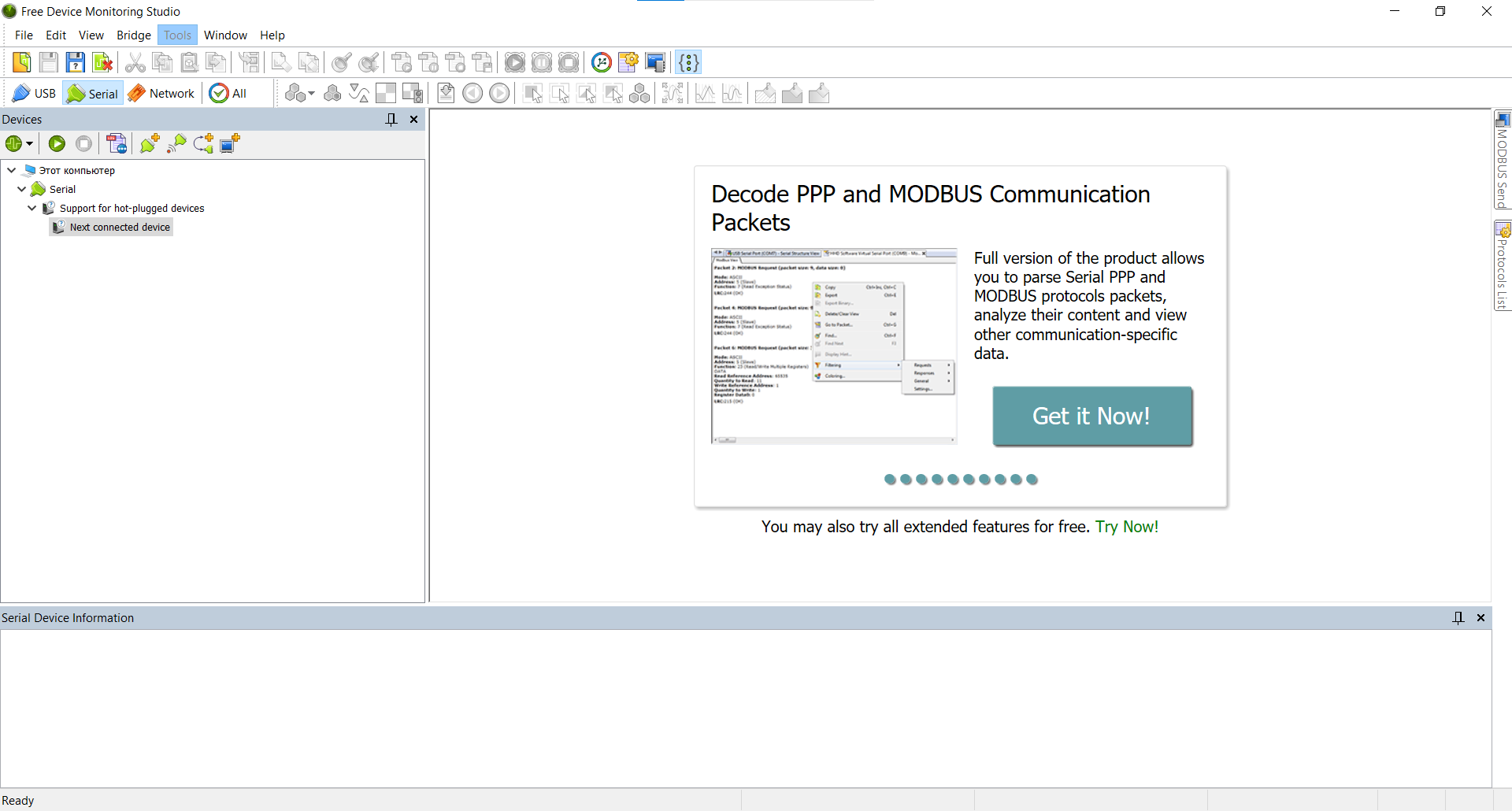


Рисунок 1.12 – дизайн ПО Device Monitoring Studio

**Наполненность интерфейса**

При запуске приложения пользователю открывается окно, разделенное на 4 блока:

* блок взаимодействия с сессией и протоколами (рисунок 1.13);



Рисунок 1.13 – блок взаимодействия с сессией

* блок взаимодействия со всеми подключенными устройствами и отправленными/полученными пакетами данных (рисунок 1.14);

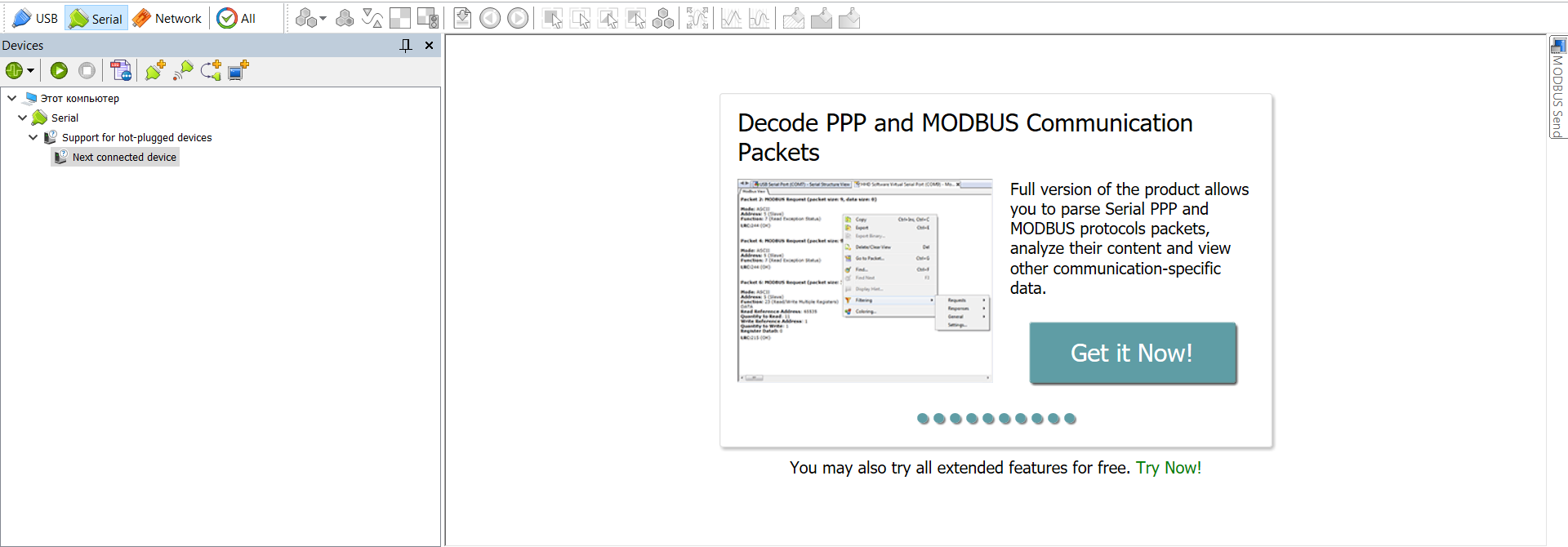


Рисунок 1.14 – блок взаимодействия со всеми подключенными устройствами

* блок, информирующий о состоянии подключенного устройства (рисунок 1.15);



Рисунок 1.15 – блок с информацией об устройстве

* блок информирования о состоянии подключения (рисунок 1.16).



Рисунок 1.16 – блок информирования о состоянии подключения

**Доступность использования**

ПО Device Monitoring Studio является платным и распространяется в 3 вариантах для личного использования:

* Standard версия стоимостью $164.99;
* Professional версия стоимостью $374.99;
* Ultimate версия стоимостью $499.99.

Но также есть пробный период на 2 недели, в течение которого можно протестировать ПО и решить, стоит его приобретать или нет.

**Предоставляемые функции Device Monitoring Studio**

* подключение к устройству по COM-порту. Требуется выбрать в блоке подключенных устройств тип подключения (Ethernet, USB, COM) требуемый COM-порт;
* сессии подключений. После выполнения определенной работы можно сохранить процесс, чтобы вернуться к определенному моменту после следующего запуска программы;
* состояние подключения. После подключения к выбранному COM-порту в нижнем блоке приложения будет отображаться подключено устройство или нет;
* создание виртуального COM-порта. При отсутствии требуемого адаптера есть возможность протестировать ПО с помощью виртуального COM-порта;
* создание «моста» между подключенными устройствами. Можно обмениваться данными между двумя уже подключенными устройствами.
* подключение к серверу. Можно подключиться к серверу с существующими виртуальными COM-портами;
* выбор формата вывода полученных данных. Есть возможность отображения приходящих с подключенного устройства данных;
* выбор палитры цветов отображения полученных данных. При большом количестве получаемых пакетов данных есть возможность отображать разными цветами данные от разных устройств.

**Вывод**

На рынке специализированного ПО для настройки и диагностики оборудования через COM-порт Device Monitoring Studio является самым распространенным обеспечением. Являясь самым дорогим ПО на рынке, Device Monitoring Studio предоставляет самый обширный функционал для работы с подключенными устройствами (даже с возможностью выбора интерфейса подключения).

**1.2 Вывод по анализу и формирование требований**

Из рассмотренных аналогов наиболее подходящим для сравнения с разрабатываемым дипломным проектом является ПО Titan CAN Test, т. к. оно предоставляет как минимальный функционал, так и возможность довольно тонкой настройки и диагностики оборудования через COM-порт. Так же это ПО является простым в освоении и для новичка, и для профессионала.

В рамках дипломного проекта поставлена цель разработать настольное приложение «ПО настройки и диагностики отдельных узлов приемо-передающего модуля». Для этого установлены минимальные требования к проекту:

* приложение должно соответствовать MVVM-паттерну;
* для хранения данных должна быть использована компактная база данных SQLite;
* программа должна иметь понятный пользователю интерфейс с возможностью точной настройки подключения к COM-порту;
* реализовать набор команд по разработанному протоколу;
* реализовать возможность диагностики подключенных по интерфейсу CAN устройств;
* реализовать разбор ошибок, получаемых от подключенных устройств;
* реализовать формирование CAN-сообщений;
* реализовать разбор сообщений, получаемых от подключенных устройств;
* реализовать формирование пакета данных для передачи по интерфейсу CAN;
* реализовать возможность просмотра полученных от подключенных устройств данных.

Для достижения поставленных целей требуется решить требуемые задачи:

* изучить аналогичные ПО, представленные на рынке настройки и диагностики оборудования
* изучить необходимые технологии для реализации дипломного проекта;
* спроектировать архитектурное решение настольного приложения;
* разработать ПО на заявленную тему;
* протестировать разработанный в дипломном проекте функционал;

**2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ**

**2.1 Концептуальные сведения**

**2.1.1** Настольное приложение

Настольное приложение – программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы. Такие приложения высокопроизводительные, могут работать напрямую с принтерами, сканерами, факсами и прочей техникой.

Плюсы настольного приложения перед веб-приложением:

* автономная работа как с подключением, так и без подключения к интернету;
* всегда быстро запускается, т. к. данные загружаются из памяти устройства, на котором установлено, не требует каждый раз загрузки параметров из сети;
* работа с периферийными устройствами, подключенными на данный момент к компьютеру [1].

**2.1.2** Монолитная архитектура

Монолитная архитектура — это вычислительная сеть с большой кодовой базы, внутри которой решаются бизнес-задачи. Чаще всего монолитной архитектурой описываются приложения, которые тяжело масштабировать, из-за чего серьезно усложняется процесс обновления приложения.

В сравнении монолитной ставится микросервисная архитектура. Микросервисная архитектура – это способ организации архитектуры приложения, при котором независимо друг от друга развертываются несколько приложений, каждое из которых имеет собственную кодовую базу. Такой подход существенно упрощает процесс масштабирования приложения, т. к. при обновлении, осуществляются изменения лишь в одном отдельном приложении.

В разрабатываемом дипломном проекте было выбрана монолитная архитектура, т. к. не требуется развертывание отдельных приложений и баз данных, т. к. приложение является однопользовательским и для этого не требуются лишние затраты на ресурсы [2].

**2.1.3** MVVM

Model-View-ViewModel – шаблон проектирования, позволяющий отделить модель от ее представления, чтобы они изменялись независимо друг от друга. Этот шаблон хорошо подходит для технологий, где применяется концепция «связывания данных», позволяет легко расширять функционал приложения [3].

**2.1.4** Язык программирования C#

C# – объектно-ориентированный язык программирования общего назначения со статической типизацией, поддерживает событийное программирование и анонимные функции с замыканиями, LINQ и XML комментарии.

Этот язык программирования был разработан в период с 1998 по 2002 год командой инженеров, во главе которой стояли Андерс Хейлсберг и Скотт Вильтаумот. Т.к. в свое время Андерс Хейлсберг работал над языком Delphi, то в следующей его разработке – языке программирования C# он многое позаимствовал именно от Delphi.

Также, хоть C# и называют «Java от Microsoft», но разработчики C# все же исключили достаточно много проблемных и нелогичных практик при разработке своего языка, например множественное наследование классов.

Язык имеет C-подобный синтаксис, сильно приближенный к языкам программирования C++ и Java и имеет следующие особенности:

* статическая типизация;
* перегрузка операторов;
* анонимные функции и делегаты;
* атрибуты, события;
* обобщенные функции.

C# быстро занял свою часть рынка разработки программного обеспечения и по сей день используется в таких сферах как:

* игровая индустрия, а именно игровой движок Unity, применяемый в разработке приложений с 2D-графикой и 3D-графикой, конкурируя с C++ и его движком Unreal Engine;
* веб-разработка, в частности разработка серверной части приложений, но также может применяться и для разработки клиентской части;
* разработка настольных приложений, где он является несомненным лидером;
* разработка мобильных приложений с применением фреймворка Xamarin.Forms, а начиная с мая 2022 года Microsoft ввели новую технологию называемую MAUI, которая может применяться для кроссплатформенной разработки мобильных и десктопных приложений [4].

**2.1.5** WPF

WPF – система построения настольных приложений, в основе которой лежит векторная система визуализации, не зависящая от расширения устройства вывода.

Для начала требуется рассмотреть потомка WPF, а именно систему Windows Forms, которая была включена в набор .NET с самого его появления.

Windows Forms быстро стала предпочитаться разработчиками пользовательского интерфейса из-за своей мощной объектной модели, но при том простого механизма построения интерфейсов. Хоть с помощью этой системы и было построено огромное множество пользовательских интерфейсов простых и сложных проектов, стоит отметить, что её программная модель не идеальна и ассиметрична, т.е. сборки System.Windows.Forms.dllиSystem.Drawing.dll не обеспечивают прямой поддержки многих технологий для построения пользовательских интерфейсов. Например, Windows Forms не охотно позволяет разработчику создать собственный элемент управления пользовательского интерфейса, т.к. полностью основан на API-интерфейсе Windows или проще говоря WinAPI.

Исходя из недостатков Windows Forms начиная с .NET 3.0 Microsoft ввели новую библиотеку для построения настольных приложений Windows Presentation Foundation или же просто WPF.

В WPF ввели новую модель с совершенно другим устройством и подходом к построению приложений. Хоть в WPF и имеются очень знакомые элементы управления, но каждый из них эта библиотека рисует самостоятельно, что позволяет ей предоставить разработчику легко изменять для своей задачи любой элемент управления пользовательским интерфейсом и для этого может и не потребоваться написание кода на C#, а достаточно будет изменить элемент с помощью языка разметки XAML.

Microsoft при разработке WPF перешли на использование мощного API-интерфейса графики с аппаратным ускорением DirectX, используемый в разработке современных видеоигр, что позволяло использовать серьезные графические эффекты при этом не платя за это ресурсами компьютера.

В этой системе реализована концепция «связывания данных», что с самого начала разработки приложения предлагает использовать паттерн MVVM. В маленьких проектах это может привести к некоторым трудностям, в отличии от прямого подхода к построению таких приложений, но стоит проекту начать хоть немного разрастаться и MVVM паттерн начнет показывать всю мощь такого подхода к построению пользовательских интерфейсов [5].

**2.1.6** SQLite

SQLite – это быстрая и легкая встраиваемая СУБД, состоящая из одного файла на языке C, которая не имеет сервера и позволяет хранить всю базу локально на одном устройстве.

Встраиваемая СУБД – СУБД, не использующая архитектуру клиент-сервер. Движок является не отдельно работающим процессом, с которым взаимодействует программа, наоборот, программа компонуется с ней, а движок служит составной частью программы.

Преимущества СУБД SQLite:

* не требует отдельного процесса сервера;
* не требуется отдельное администрирование и настройка такой базы данных, т.к. она поставляется в нулевой конфигурации;
* очень легковесна, т.к. полностью настроенная база данных весит не более четырехсот килобайт, а файл с базовой настройкой и опущенными ненужными функциями займет на диске не более двухсот пятидесяти килобайт;
* автономна, что значит отсутствие внешних зависимостей;
* поддерживает абсолютное большинство запросов, описанных в стандарте SQL92;
* написана на языке программирования ANSI-C с простым и понятным API для использования;
* доступна как в UNIX-системах, так и в Windows.

В качестве протокола обмена данными используется API SQLite.

В разрабатываемом дипломном проекте СУБД SQLite является идеальным вариантом, т.к. разрабатываемое ПО является однопользовательским и не требуется дорогостоящая поддержка сервера с базой данных [6].

**2.1.7** COM-порт

COM-порт – двунаправленный последовательный интерфейс, предназначенный для обмена байтовой информацией между устройствами. Последовательным называется, т. к. информация, выраженная в битах, передается друг за другом. Ранее использовался для подключения терминала, затем для модема и мыши. На рисунке 2.1 приведен пример современного варианта COM-порта.

Изображение выглядит как металл, машина, коробка передач, серебряный

Автоматически созданное описание

Рисунок 2.1 – пример COM-порта

На рисунке 2.2 приведен один из вариантов подключения COM-порта к компьютеру для обмена данными с модемом.

Изображение выглядит как ноутбук, компьютер, электроника, соединитель

Автоматически созданное описание

Рисунок 2.2 – вариант подключения COM-порта

В данном дипломном проекте технология обмена данными через интерфейс RS-232 используется для обмена данными между аппаратным устройством и компьютером [7].

**2.1.8** CAN

CAN – стандарт промышленной сети, ориентированный, прежде всего, на объединение в единую сеть различных исполнительных устройств и датчиков. Один из режимов работы – последовательный, т. е. соответствует режиму работы COM-порта, что предполагает их использование в паре. Чтобы абстрагироваться от среды передачи данных, т. е. от физического уровня, спецификация CAN не предполагает описание битов как «0» и «1», а как рецессивный и доминантный. При доминантном бите сигнал есть, а при рецессивном сигнала нет и стандарт CAN требует от физического уровня только одно условие, чтобы доминантный бит мог подавить рецессивный, но никак не наоборот.

Ключевые характеристики CAN-шины:

* Витая пара проводов и шлейф являются бюджетным вариантов для организации CAN-шины, но из-за этого показывают низкую скорость передачи данных. При использовании оптоволокна можно обеспечить высокую надежность и долгую работоспособность, а также высокую скорость передачи данных;
* Практически все CAN-шины можно разделить на CAN2 0A, которая умеет принимать пакеты данных в формате одиннадцати бит, но не сможет обработать ошибки, переданные от микроконтроллера и CAN2 0B, умеющая принимать как пакеты диагностических данных, так и пакеты с ошибками микроконтроллеров.

На рисунке 2.3 показан пример работы CAN-шины, где CAN-high – это CAN2 0B, а CAN-low – это CAN2 0A [8].

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 2.3 – пример работы CAN-шины

**3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

**3.1 Общие сведения**

После рассмотрения узлов разрабатываемого программного обеспечения был выделен список функциональных требований и выбрана монолитная архитектура для реализации правильно построенной системы.

Монолитная архитектура – это традиционная модель программного обеспечения, которая представляет собой единый модуль, работающий автономно и независимо от других приложений.

В то же время разрабатываемое ПО разделено на программные модули:

* модуль формирования пакета данных CAN-шины;
* модуль взаимодействия с базой данных;
* модуль графического интерфейса.

**3.2 Модуль взаимодействия с базой данных**

Этот модуль полностью инкапсулирует взаимодействие с базой данных SQLite.

База данных в разрабатываемом ПО используется только для хранения данных о опрашиваемых аппаратных устройствах, поэтому представляет собой одну сущность, описанную в таблице 3.1.

Таблица 3.1 – Device

|  |  |  |
| --- | --- | --- |
| Название колонки | Тип колонки | Описание |
| Id | INTEGER | Идентификатор устройства |
| Name | VARCHAR(100) | Название устройства |
| Type | VARCHAR(100) | Тип устройства |
| Address | INTEGER | Адрес устройства |

**3.3 Модуль графического интерфейса**

Модуль графического(пользовательского) интерфейс представлен WPF-приложением, которое позволяет пользователю легко взаимодействовать с аппаратными устройствами, не прибегая к программному обеспечению, которое требует напрямую задавать байты определенных команд.

В разрабатываемом ПО пользователю всего лишь требуется выбрать COM-порт, по которому подключено оборудование, тип оборудования с соответствующим адресом и команду, которую требуется передать на выбранное устройство, затем нажать кнопку отправления запроса. Если команда подразумевает получения ответа от аппаратуры, то в отдельной таблице пользователь увидит ответ устройства, также возникающие ошибки при работе с подключенными устройствами.

**3.4 Модуль формирования пакета данных CAN-шины**

На этом этапе происходит формирование пакета данных, передаваемого по COM-порту на аппаратное устройство. В соответствии с протоколом пакет данных CAN-шины соответствует определенному формату (таблица 3.1):

Таблица 3.1 – формат пакета CAN-шины

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поле | Приоритет | Команда | Код устройства | Адрес устройства |
| К-во бит | 2 | 4 | 3 | 20 |
| Значение | 0 … 3 | 0 … 15 | 0 … 7 | 0 … 1048575 (0xFFFFF) |

В протоколе выделяются несколько видов опрашиваемых аппаратных устройств (таблица 3.2):

* контроллер;
* синтезатор гетеродина с фиксированной частотой;
* синтезатор гетеродина с перестраиваемой частотой;
* приёмные модули;
* приемо-передающие модули.

Таблица 3.2 – виды устройств

|  |  |  |  |
| --- | --- | --- | --- |
| Код устройства | Адрес устройства | Тип устройства | Расшифровка |
| 0 | 0 (не адресуется) | CTRL | Контроллер |
| 1 | 0 (не адресуется) | LO1 | Синтезатор гетеродина с фиксированной частотой |
| 2 | 0 (не адресуется) | LO2 | Синтезатор гетеродина с перестраиваемой частотой |
| 3 | xx xxxx … xx xxxx | RX | Приёмные модули |
| 4 | xx xxxx … xx xxxx | TX-RX | Приёмо-передающие модули |
| 5, 6 | – | – | Не используется |
|  |  |  |  |
| 1…4 | 0xFFFFF | – | Широковещательное сообщение всем устройствам с конкретным кодом / типом |
| 0x7 | 0xFFFFF | – | Широковещательное сообщение всем устройствам на шине. |

В соответствии с протоколом, команды подразделяются на 5 видов:

* настройка устройства (таблица 3.3);

Таблица 3.3 – команда настройки устройства

| ID | Название | Объём данных | Байты данных |
| --- | --- | --- | --- |
| 0 | Настройка устройства | 8 | 1) Команда начала инициализации / сброса устройства  0xA5 (reset / сброс), после получения значения;  2) 0x0F – отключить коррекцию коэффициента усиления;  3) 0x0F – отключить температурную коррекцию коэффициента усиления передатчика;  4) не используется;  5) не используется;  6) не используется;  7) Установить линию TEST 1 (0 / 1);  8) Установить линию TEST 2 (0 / 1);  *Установите 0xFF (VAL NOT AVAILABLE) если нет необходимости изменять значение байта.* |

* рабочие параметры устройства (таблица 3.4);

Таблица 3.4 – команда рабочих параметров устройства

| ID | Название | Объём данных | Байты данных |
| --- | --- | --- | --- |
| 1 | Рабочие параметры устройства | 6 | 1)  Функционирование устройства:  0 – запретить – состояние после запуска;  1 – разрешить – разрешить функционирование устройства и включение выходной мощности, основной режим работы;  2 – разрешить работу приёмника (RX), выключить передатчик (TX).  *Примечание: В режиме работы 2 устройства «TX-RX» функционируют как «RX». Происходит отключение предварительных усилителей передающего тракта (POWER TX 8V = 0, POWER TX 3V = 0, SW TX1 ENA = 0 и SW TX2 ENA = 0). Выходной ключ SPDT TX/nRX продолжает работать в соответствии с временной диаграммой для обеспечения защиты приёмного тракта от других устройств «TX-RX», когда они переключаются в режим передачи.*  2)  Рабочая частота комплекса.  Диапазон: 140 – 240 МГц, ЦМР = 1 МГц.  Подстройка фазы передающего тракта.  3)  Младший байт фазы;  4)  Старший байт фазы.  Диапазон: 0…359°, ЦМР = 1°.  *Примечание: Только для устройств типа «TX-RX».*  5)  Уровень ослабления, устанавливаемый на аттенюатор приёмного тракта.  Диапазон 0…31 дБ, ЦМР = 1 дБ.  *Примечание: Для устройств типа «TX-RX» и «RX».*  6)  Длительность нахождения устройства в состоянии «BLANK». После старта равно 25 мкс.  Диапазон 0…250 мкс, ЦМР = 1 мкс.  7…8) не используется. |

* коррекции коэффициента усиления – состоит из двух передаваемых пакетов данных (таблица 3.5);

Таблица 3.5 – команда коррекции коэффициента усиления

| ID | Название | Объём данных | Байты данных |
| --- | --- | --- | --- |
| 2 | Коррекция коэффициента усиления тракта передатчика в устройствах | 8 | Коррекция коэффициента усиления тракта передатчика в устройствах «TX-RX».  ЦМР = 0,1 дБ, **прямой код**.  Ку = ( x · 0,1 ) дБ.  Диапазон: -10 … +10 дБ.  Коррекция выходной мощности в синтезаторах гетеродинов «LO1» и «LO2»  ЦМР = 0,1 дБ, **прямой код**.  Ку = ( x · 0,1 ) дБ.  Диапазон: -3 … +3 дБ.  Байты:  1) Общий коэффициент усиления (**ОК**).  Для устройств: «TX-RX», «LO1» и «LO2».  После включения всегда равен 0.  Частотно-зависимые коэффициенты (**ЧК**):  Для устройств: «TX-RX» / «LO2».  2) Коэффициент для 140 / 760 МГц;  3) Коэффициент для 150 / 770 МГц;  4) Коэффициент для 160 / 780 МГц;  5) Коэффициент для 170 / 790 МГц;  6) Коэффициент для 180 / 800 МГц;  7) Коэффициент для 190 / 810 МГц;  8) Коэффициент для 200 / 820 МГц;  *Установите 0xFF (VAL NOT AVAILABLE) если нет необходимости изменять значение байта.* |
| 3 |  | 4 | 1) Коэффициент для 210 / 830 МГц.  2) Коэффициент для 220 / 840 МГц;  3) Коэффициент для 230 / 850 МГц;  4) Коэффициент для 240 / 860 МГц;  5-8) не используется.  После запуска частотно-зависимые коэффициенты устанавливаются в откалиброванное значение или 0.  *Установите 0xFF (VAL NOT AVAILABLE) если нет необходимости изменять значение байта.* |

* запрос параметра (таблица 3.6).

Таблица 3.6 – команда запроса параметра

| ID | Название | Объём данных | Байты данных |
| --- | --- | --- | --- |
| 14 | Запрос параметра | 2 | 1) ID запрашиваемого параметра;  2) Номер параметра состояния устройства (используется только для параметра с ID = 15). |

В соответствии с таблицей 3.6 при передаче ID = 15 в качестве первого байта, требуется также указать номер запрашиваемого состояния в качестве второго байта. Состояний, которых можно вычитать из аппаратного устройства, в соответствии с протоколом может быть 9 вариантов, которые приведены в таблице 3.7.

Таблица 3.7 – состояния устройства

| ID | № | Название | Объём данных | Байты данных |
| --- | --- | --- | --- | --- |
| 15 | 0 | Состояние устройства | 2…8 | 1) № параметра = 0;  2…8) Код состояния устройства, смотри таблицу **Ошибка! Источник ссылки не найден.**.  В случае, если нет ошибок то передаётся пакет с 2 байтами данных [ 0, 0 ];  Иначе в одном пакете передаётся до 7 зарегистрированных ошибок. |
| 15 | 1 | Параметр диагностики 1  DC-DC | 8 | 1) № параметра = 1;  2) Температура БП, ЦМР=1°, прямой код;  3) 30 В, ЦМР = 0,25 В;  4) 5,5 В (процессор), ЦМР = 0,025 В;  5) 12 В, ЦМР = 0,1 В;  6) 50 В вход, ЦМР = 0,25 В;  7) 50 В питание УМ, ЦМР = 0,25 В;  8) не используется (0xFF). |
| 15 | 2 | Параметр диагностики 2  System | 8 | 1) № параметра = 2;  2) 3,3 В, ЦМР = 0,025 В;  3) 5,0 В, ЦМР = 0,025 В;  4) Температура ЦП, ЦМР=1°, прямой код;  5) Плис загружена (1/0) (RX , TX-RX);  6) Время загрузки ПЛИС, мс (RX, TX-RX);  7) Состояние линии TEST 1 (0 / 1);  8) Состояние линии TEST 2 (0 / 1). |

Продолжение таблицы 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | № | Название | Объём данных | Байты данных |
| 15 | 3 | Параметр диагностики 3  для устройств  RX-TX, RX | 8 | 1) № параметра = 3;  2) 1,2 В, ЦМР = 0,025 В (RX, TX-RX);  3) 2,5 В, ЦМР = 0,025 В (RX, TX-RX);  4) 3,0 В, ЦМР = 0,025 В (RX, TX-RX);  5) 5,0 В, ЦМР = 0,025 В (RX, TX-RX);  6) 3,0 В, ЦМР = 0,025 В (TX-RX);  7) 8,0 В, ЦМР = 0,050 В (TX-RX);  8) Температура усилителя, ЦМР = 1°, прямой код (TX-RX). |
| 15 | 4 | Параметр диагностики 4  для устройств  LO1, LO2 | 4 | 1) № параметра = 4;  2) Захват частоты (0 / 1) (LO1, LO2);  3) **Наличие** входной мощности (0 / 1) (LO1, LO2);  4) Состояние опорного генератора для LO1 (нормальное состояние 0x00h):  младшие 4 бита [0..3]:  0 — нормальный режим, нет частоты;  1 — нормальный режим, нет захвата;  2 — нормальный режим, захват;  3 — режим трансляции, нет частоты;  4 — режим трансляции, есть частота;  5 — автозахват.  [4] бит — состояние питания:  0 — напряжение питания в норме;  1 — сбой электропитания. |

Продолжение таблицы 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | № | Название | Объём данных | Байты данных |
| 15 | 5 | Параметр диагностики 5  для устройств TX/RX, частично RX | 8 | 1) № параметра = 5;  2) Уровень мощности гетеродина LO1,  ЦМР = 0,1 дБ. P = 0 дБм + x \* 0,1 дБ;  (TX-RX, RX);  3) Уровень мощности гетеродина LO2,  ЦМР = 0,1 дБ. P = 0 дБм + x \* 0,1 дБ;  (TX-RX, RX);  4) Значение, установленное на электрически-управляемый аттенюатор (**А**) передающего тракта (TX). ЦМР = 0,1 дБ..  Уровень выходной мощности,  ЦМР = 0,1 дБ. P = 0 дБм + x \* 0,1 дБ;  (TX-RX);  5) Младший байт;  6) Старший байт;  Уровень отражённой мощности,  ЦМР = 0,1 дБ. P = 0 дБм + x \* 0,1 дБ;  (TX-RX);  7) Младший байт;  8) Старший байт; |

Продолжение таблицы 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | № | Название | Объём данных | Байты данных |
| 15 | 6 | Параметр диагностики 6  для устройств  TX/RX, RX | 6 | 1) № параметра = 6;  2) резерв;  Время от прихода сигнала R0 = 0 до перехода в состояние TX.  T(R0→TX) = T(BLANK ON) + T(BLANK) + T(RXoff) + T(TXon);  3) Младший байт;  4) Старший байт;  Время от прихода сигнала R0 = 1 до перехода в состояние RX.  T(R0→RX) = T(TXoff) + T(RXon);  5) Младший байт;  6) Старший байт;  ЦМР = 0,01 мкс для всех значений.  *Примечание: Для устройств типа «TX/RX» и «RX»;*  *Описание функционирования приведено в документе «Временные диаграммы»* |
| 15 | 7 | Параметр диагностики 7  для устройств  TX/RX, RX | 8 | 1) № параметра = 7;  2) резерв;  Время исполнения T(BLANK ON);  3) Младший байт;  4) Старший байт;  Время исполнения T(BLANK);  задаётся в параметре 1, байт 6.  5) Младший байт;  6) Старший байт;  Время исполнения T(RXoff);  7) Младший байт;  8) Старший байт;  ЦМР = 0,01 мкс для всех значений.  *Примечание: Для устройств типа «TX-RX» и «RX»;*  *Описание функционирования приведено в документе «Временные диаграммы»* |

Продолжение таблицы 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | № | Название | Объём данных | Байты данных |
| 15 | 8 | Параметр диагностики 8  для устройств  TX/RX, RX | 8 | 1) № параметра = 8;  2) резерв;  Время исполнения T(TXon);  7) Младший байт;  8) Старший байт;  Время исполнения T(TXoff);  5) Младший байт;  6) Старший байт;  Время исполнения T(RXon);  7) Младший байт;  8) Старший байт;  ЦМР = 0,01 мкс для всех значений.  *Примечание: Для устройств типа «TX-RX» и «RX»;*  *Описание функционирования приведено в документе «Временные диаграммы»* |

**4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

**4.1 Формирование пакета данных CAN-шины**

**4.1.1** Конфигурация подключения по COM-порту

Для разработки программного обеспечения был выбран язык программирования C#. Для настройки подключения по COM-порту был использован класс SerialPort из библиотеки System.IO.Ports. Формирование подключения происходит в конструкторе класса, выполняющего обмен данными по COM-порту (приложение А). Задаются такие условия подключения как:

* выбранный пользователем порт;
* частота передачи данных, заданной в родительском классе;
* время на запись/чтение данных по COM-порту, которые также заданы в родительском классе;
* значение поля, которое позволяет отправлять сигнал на аппаратное устройство, подключенное по COM-порту.

**4.1.2** Формирование пакета данных команды настройки устройства

Класс SetupDeviceCommand (приложение А) унаследован от одноименного интерфейса для реализации инъекции зависимости. В классе содержится пять публичных функций:

* получение параметров сброса устройства после получения начальных значений;
* получение параметров отключения коррекции коэффициента усиления;
* получение параметров отключения температурной коррекции коэффициента усиления;
* получение параметров установления значения линии TEST 1;
* получение параметров установления значения линии TEST 2.

Каждая из этих функций формирует байтовую строку для последующей передачи на контроллер и изменения одного конкретного байта данных.

Также любая из этих функций возвращает кортеж, состоящий из 3 параметров, т.к. требуется получить 3 переменных, которые содержат данные о команде:

* id – параметр, который содержит информацию о приоритете команды, её номере, код и адрес устройства которому пользователь отправил данную команду;
* data – строка, содержащая передаваемые данные – байты, которые по протоколу, могут изменять поведение аппаратного устройства;
* dataVolume – объем передаваемых данных (количество передаваемых байт в параметре data).

Классы, реализующие другие команды, определенные в протоколе, спроектированы аналогичным образом.

### **4.1.3** Формирование пакета данных команды запроса параметров

Класс RequestParametersCommand (приложение А) реализует одноименный интерфейс для возможности использования инъекции зависимостей. В этом классе реализована одна функция, которая, как и в подразделе, описанном выше, возвращает кортеж из трех параметров.

Эта функция принимает одним из параметров Id вызванной команды. В зависимости от этого параметра функция формирует строку с данными, т.к. может быть вызвана команда на вычитывание данных с состоянием или же без него.

## **4.2 Чтение и запись данных по COM-порту**

В классе, описанном в подразделе выше, есть две приватных переменных, которые упоминались в первом разделе данной главы и публичное поле, которое запоминает отправленное сообщение для вывода на экран пользователя при отправке запроса (рисунок 4.1).

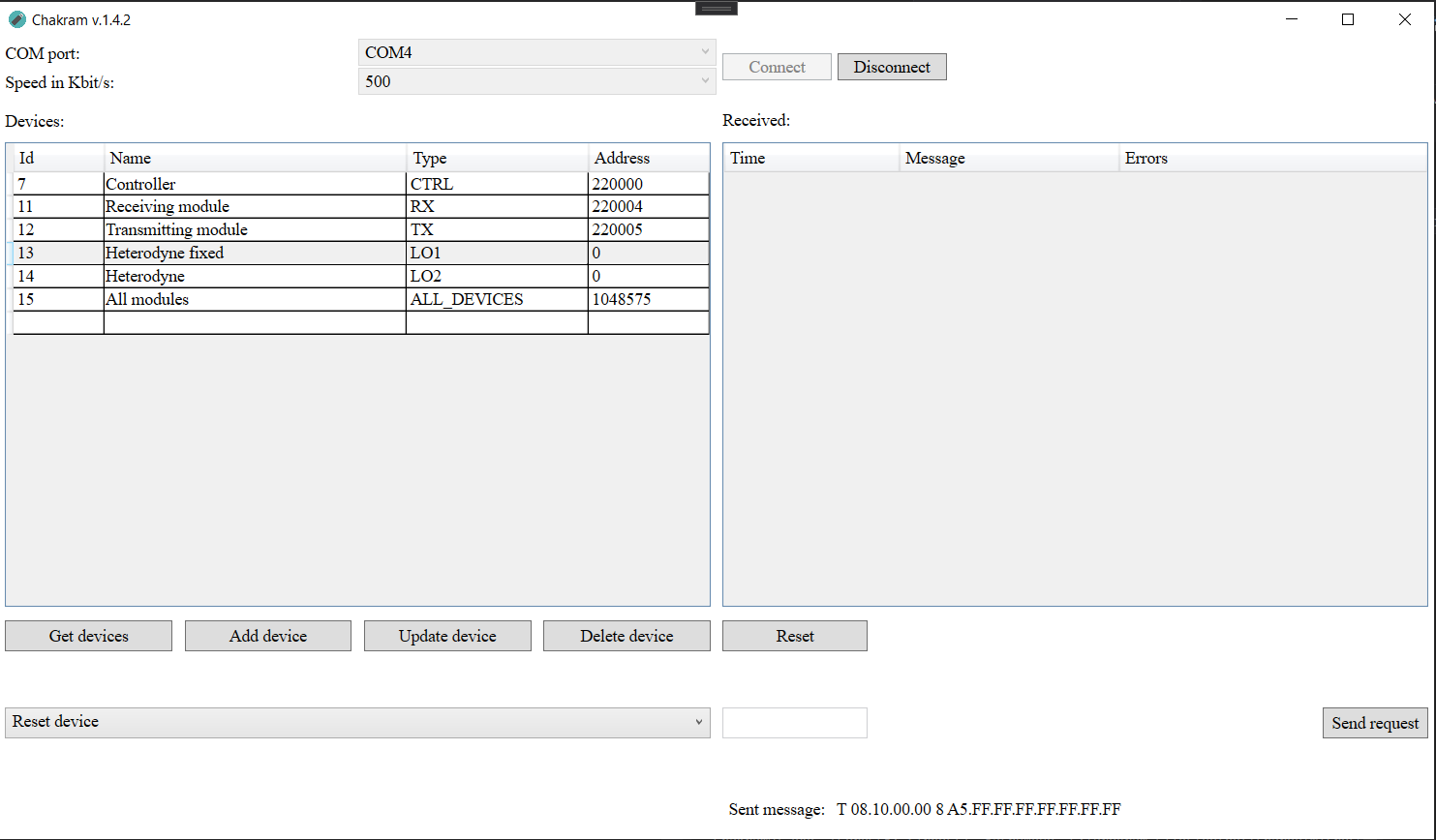


Рисунок 4.1 – отображение формата отправленной команды

Также в классе есть два приватных метода Connect() и Disconnect() используемые для подключения и отключения от COM-порта соответственно и два публичных метода WriteDataPacket() и ReadDataPacket() для записи и чтения данных из COM-порта соответственно.

Функция ReadDataPacket() при открытом подключении к COM-порту выполняет побайтовое считывание данных и записывает в очередь, состоящую из байтов, которую затем и возвращает из функции.

Функция WriteDataPacket() принимает на вход параметры, содержащие информацию об отправленной команде, данные и объем передаваемых данных. Затем функция преобразовывает их в удобный вид для чтения человеком, выводит в консоль (для отладки) и записывает данные в COM-порт.

## **4.3** **Формирование протокола передачи данных по COM-порту**

Класс ChakramProtocol (приложение А) реализует интерфейс IDisposable для возможности использования его объектов с конструкцией using() и последующим очищением управляемых и неуправляемых ресурсов, для этого также используются функция Dispose() и деструктор.

Перед конструктором класса объявлено четыре приватных переменных типами интерфейсов, определяющих команды протокола. Далее эти переменные проинициализированы в конструкторе для реализации инъекции зависимостей.

Первые тринадцать функций, определенных в классе, формируют протокол для обмена данными между клиентской частью и аппаратными устройствами.

Работают эти функции схожим образом:

* вызов соответствующей команде функции и получение требуемых параметров для передачи по COM-порту;
* запись полученных данных в COM-порт с помощью реализованного класса CANController.

## **4.4 Модуль взаимодействия с базой данных**

Взаимодействие с базой данных реализовано с помощью библиотеки Dapper, которая позволяет легко получать и записывать данные в базу данных, так же легко обрабатывать полученные данные в C#. Выполняются эти действия с помощью описанных программистом вручную SQL-запросов.

За взаимодействие с базой данных в разрабатываемом дипломном проекте отвечает класс DeviceRepository (приложение А), в котором находится 5 публичных асинхронных методов:

* получение всех устройств из базы данных;
* получение одного устройства из базы данных по переданному параметру Id;
* добавление нового опрашиваемого устройства;
* обновление выбранного устройства;
* удаление выбранного опрашиваемого устройства.

Каждая из этих функций открывает подключение к базе данных с помощью строки подключения, заданной в отдельном классе, содержащем конфигурации базы данных. Затем каждая функция исполняет требуемый запрос и возвращает полученные данные, если требуется.

Также этот класс реализует одноименный интерфейс для реализации инъекции зависимостей.

## **4.5 Клиентская часть**

Клиентская часть реализована с помощью системы для построения настольных приложений WPF, которая для использования своего механизма связывания данных требует соответствие с MVVM-паттерном, который имеет определенную структуру (рисунок 4.2).

Изображение выглядит как текст, снимок экрана, Шрифт, белый

Автоматически созданное описание

Рисунок 4.2 – паттерн MVVM

В разрабатываемом ПО View представляет собой оконный интерфейс, разрабатываемый на языке XAML (приложение А). На рисунке 4.3 показана часть приложения для взаимодействия пользователя с базой данных.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.3 – взаимодействие пользователя с базой данных

Класс ViewModel для взаимодействия с базой данных реализован на языке программирования C# (приложение А) и требуется для взаимодействия уровней View и Model.

И так же уровень Model, который содержит модель, с которой происходит вся работа для ее обновления и отображения на уровне View. Класс модели Device приведен в приложении А.

**4.6 Внедрение зависимостей**

Dependency injection – процесс предоставления внешней зависимости программному компоненту. Использование данного подхода в программировании позволяет разработчику:

* сформировать хорошую основу для правильной архитектуры приложения;
* возможность повторного использования кода;
* простота рефакторинга кода;
* легкость тестирования кода.

В разрабатываемом программном обеспечении внедрение зависимостей начинается с класса App (приложение А).

В конструкторе класса App инициализируется переменная AppHost. В этой переменной регистрируются все интерфейсы, созданные в процессе разработке и классы, в конструктор которых требуется передать эти интерфейсы. В дальнейшем это позволяет при создании объекта класса не передавать вручную несколько интерфейсом, а создать объект с помощью, проинициализированной при запуске приложения переменной AppHost.

Пример создания объектов с помощью внедрения зависимостей приводится в функциях OnStartup() и OnExit() класса App.

Функция OnStartup() получает из переменной AppHost класс главного окна MainWndow и затем устанавливает контекст данных, также полученный из зарегистрированных сервисов. Также эта функция с помощью заданных сервисов задает первоначальную инициализацию базы данных.

Функция OnExit() получает класс ViewModel из сервисов переменной AppHost и проверяет перед закрытием главного окна приложения, нажата ли кнопка «Disconnect» для отключения от COM-порта, чтобы не оставлять открытых подключений после завершения работы.

# **5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

В этой главе будет описано тестирование функциональных модулей программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля. Здесь будут рассматриваться такие функции как:

* подключение к COM-порту;
* отключение от COM-порта;
* получение/добавление/обновление/удаление опрашиваемых устройств;
* отправление команды протокола на опрашиваемое устройство.

Тестирование вышеперечисленных функциональных модулей приведены в таблице 5.1.

Таблица 5.1 – Функциональные тест-кейсы программного продукта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования: 12.04.2023 | Тестировал: Кондрашов И.Д. | ОС:Windows 10 | Приложение:  Программное обеспечение настройки и диагностики отдельных узлов приемо-передающего модуля |  |
| Идентификатор | Модуль | Описание теста | Ожидаемый результат | Статус («выполнено успешно», «выполнено с ошибкой») |
| Т01 | Приложение не запущено | 1. Запустить в папке с приложением файл Chakram.sln | 1. Отображается главное окно приложения | Выполнено успешно |
| Т02 | Главное окно приложения | 1. Провести визуальный осмотр содержимого окна приложения;  2. Изменить размер окна приложения; | 1. Все элементы окна отображаются корректно, грамматические ошибки отсутствуют;  2. Элементы окна отображаются корректно;  3. Главная страница приложения не отображается, отображение каталога автозапчастей; | Выполнено успешно |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Идентификатор | Модуль | Описание теста | Ожидаемый результат | Статус («выполнено успешно», «выполнено с ошибкой») |
| Т03 | Подключение к COM-порту | 1. Нажать на выпадающий список, который располагается напротив метки «COM port»;  2. Нажать на выпадающий список, который располагается напротив метки «Speed in Kbit/s»;  3. Нажать на кнопку «Connect»; | 1. Отображаются все устройства, подключенные по COM-порту;  2. Отображаются доступные скорости передачи данных по COM-порту;  3. Отображается сообщение об успешном подключении к выбранному COM-порту; | Выполнено успешно |
| Т04 | Получение опрашиваемых устройств | 1. Нажать на кнопку «Get devices»; | 1. При наличии устройств в таблице базы данных они отобразятся в таблице «Devices»; | Выполнено успешно |
| Т05 | Добавление опрашиваемых устройств | Предварительное условие: выбрано устройство в таблице «Devices».  1. Нажать на кнопку «Add device»;  2. В открывшемся окне ввести все валидные данные и нажать кнопку «Submit»; | 1. Отображается другое окно;  2. Отображается сообщение об успешном добавлении устройства; | Выполнено успешно |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Идентификатор | Модуль | Описание теста | Ожидаемый результат | Статус («выполнено успешно», «выполнено с ошибкой») |
| Т06 | Добавление опрашиваемого устройства | Предварительное условие: выбрано устройство в таблице «Devices».  1. Нажать на кнопку «Add device»;  2. В открывшемся окне ввести валидные имя и тип устройства, но в качестве адреса ввести не число и нажать кнопку «Submit»; | 1. Отображается другое окно;  2. Отображается сообщение о неправильном значении, введенном в поле «Address»; | Выполнено с ошибкой |
| Т07 | Изменение опрашиваемого устройства | Предварительное условие: выбрано устройство в таблице «Devices».  1. Двойным нажатием левой кнопки мыши по колонке «Type» или «Address» соответствующего устройства и изменение значения;  2. Нажать кнопку «Update device»; | 1 Изменение соответствующего значения в таблице «Devices» пользовательского интерфейса;  2. Добавление внесенных изменений в базу данных; | Выполнено успешно |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Идентификатор | Модуль | Описание теста | Ожидаемый результат | Статус («выполнено успешно», «выполнено с ошибкой») |
| Т08 | Изменение опрашиваемого устройства | Предварительное условие: выбрано устройство в таблице «Devices».  1. Двойным нажатием левой кнопки мыши по колонке «Type» или «Address» соответствующего устройства и изменение значения;  2. Не нажимать кнопку «Update device»; | 1 Изменение соответствующего значения в таблице «Devices» пользовательского интерфейса;  2. Внесенные изменения не зафиксируются в базе даных и при последующих попытках получения устройств внесенные изменения не будут отображаться; | Выполнено с ошибкой |
| Т09 | Удаление опрашиваемого устройства | Предварительное условие: выбрано устройство в таблице «Devices».  1. Нажать на кнопку «Delete device» | 1. Удаление выбранного устройства в таблице базы данных и таблице пользовательского интерфейса | Выполнено успешно |
| Т10 | Удаление полученной информации от опрашиваемых устройств | 1. Нажать на кнопку «Reset» | 1. В таблице «Received» пользовательского интерфейса удалятся все данные, полученные от опрашиваемых устройств; | Выполнено успешно |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Идентификатор | Модуль | Описание теста | Ожидаемый результат | Статус («выполнено успешно», «выполнено с ошибкой») |
| Т11 | Отправление команды протокола на опрашиваемое устройство | Предварительное условие: выбрано устройство в таблице «Device».  1. Выбрать одну из возможных команд;  2. Нажать на кнопку «Send request» | 1. В выпадающем списке с командами отображается выбранная пользователем команда;  2. В правом нижнем углу окна возле метки «Sent message» отобразится отправленное сообщение в читаемом для человека виде, также, при наличии полученных от опрашиваемого устройства сообщений, они отобразятся в таблице «Received»; | Выполнено успешно |
| Т12 | Отключение от COM-порта | 1. Нажать на кнопку «Disconnect» | 1. Отображается сообщение об успешном отключении от COM-порта | Выполнено успешно |

# **6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НАСТРОЙКИ И ДИАГНОСТИКИ ОТДЕЛЬНЫХ УЗЛОВ ПРИЕМО-ПЕРЕДАЮЩЕГО МОДУЛЯ**

## **6.1 Краткая характеристика программного продукта**

Разрабатываемое в дипломном проекте функциональное настольное приложение, позволяет пользователю точно настраивать и диагностировать отдельные узлы приемо-передающего модуля.

В рамках работы над проектом будут реализованы следующие функции:

* предоставление пользователю подключения к приемо-передающему модулю;
* настройка подключения;
* отправка команд управления;
* получение команд с ответами от устройства;
* выполнение настройки оборудования;
* осуществление диагностики;
* осуществление диагностики неисправностей;
* выполнение сервисных функций по функционированию приемо-передающего модуля.

Таким образом, вышеописанные функции позволяют пользователям приложения автоматизировать процесс настройки и диагностики приемо-передающего модуля.

Программное обеспечение создается по индивидуальному заказу Центр 1.9 НИЧ БГУИР.

## **6.2 Расчет затрат на разработку и цена программного обеспечения**

### **6.2.1** Затраты на основную заработную плату разработчиков

Для разработки данного продукта необходимо привлечь программиста, тестировщика, проектного менеджера. Команда разработки оценила трудоемкость проекта в 550 часов.

На основании данных организации-разработчика оклады специалистов команды разработки: программист - 2200 рублей/месяц, тестировщик - 1800 рублей/месяц, проектный менеджер - 2700 рублей/месяц.

Расчет основной заработной платы участников команды рассчитывается по формуле:

(6.1)

где n – количество исполнителей, занятых разработкой конкретного ПО;

- часовая заработная плата *i*-го исполнителя (руб.);

- трудоемкость работ, выполняемых *i*-м исполнителя (ч).

Исходя из вышеперечисленных данных рассчитаем затраты на основную заработную плату команды. (таблица 6.1)

Таблица 6.1 - Расчет затрат на зарплату команды разработчиков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Позиция члена команды | Вид работы | Часовой оклад, руб/ч | Трудоемкость, ч | Зарплата по тарифу, руб |
| 1 | Программист | Разработка продукта | 13,1 | 320 | 4 192 |
| 2 | Тестировщик | Тестирование продукта | 10,7 | 125 | 1 337,5 |
| 3 | Проектный менеджер | Координация процессов | 16,1 | 105 | 1 690,5 |
| Итого | | | | | 7 220 |

### **6.2.2** Затраты на дополнительную заработную плату разработчиков

Затраты на дополнительную заработную плату команды разработчиков определяется по формуле:

(6.2)

где - затраты на основную заработную плату, (руб.);

- норматив дополнительной заработной платы, 15%

### **6.2.3** Отчисления на социальные нужды

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами Республики Беларусь по формуле:

(6.3)

где ‒ норматив отчислений на социальные нужды, %. Нсоц = 34,6%.

Сумма отчислений на социальные нужды, согласно формуле (6.3), составит:

### **6.2.4** Прочие затраты

Расходы по данной статье определяются по формуле:

(6.4)

где - норматив прочих в целом по организации.

Приняв значение равным 30% и подставив в формулу (6.4) оставшиеся значения, произведем расчет прочих затрат:

### **6.2.5** Полная сумма затрат на разработку программного обеспечения

Полная сумма затрат на разработку программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля находится путем суммирования всех рассчитанных статей затрат. Расчет общей суммы затрат представлен в таблице 6.2.

Таблица 6.2 - Полная сумма затрат на разработку программного обеспечения

|  |  |
| --- | --- |
| Статья затрат | Сумма, руб. |
| Основная заработная плата команды разработчиков | 7 220 |
| Дополнительная заработная плата команды разработчиков | 1 083 |
| Отчисления на социальные нужды | 2 872,84 |
| Прочие затраты | 2 166 |
| Общая сумма затрат на разработку | 13 341,84 |

Полная себестоимость разрабатываемого программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля составит 13 341,84 рубля.

### **6.2.6** Плановая прибыль

Расчеты по данной статье исходят из затрат на заработную плату команде.

(6.5)

где - рентабельность затрат на разработку программного средства.

Приняв значение равным 12% и подставив в формулу (6.5) оставшиеся значения, произведём расчёт плановой̆ прибыли:

### **6.2.7** Отпускная цена программного средства

Расчеты по данной статье исходят из затрат на заработную плату и плановой прибыли.

(6.6)

Произведя подстановку выше полученных значений в формулу (6.6), получим отпускную цену программного средства, которая составит:

Аналогов, выполняющих полностью идентичный функционал, как и в дипломном проекте нет, что позволяет организации-разработчику свободно предлагать разрабатываемое программное обеспечение заказчику, с большой вероятностью принятия заказчиком положительного решения к покупке.

## **6.3 Оценка результата от разработки программного обеспечения**

Экономический эффект для организации-разработчика программного обеспечения заключается в продаже разработанного в дипломном проекте программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля организации-заказчику.

В результате расчет инвестиций в разработку программного обеспечения настройки и диагностики отдельных узлов приемо-передающего модуля выявлено, что его стоимость составит рублей.

Так как программное средство создается под нужды и требования конкретного заказчика и является уникальным, то экономический эффект для организации-разработчика определяется по формуле:

(6.7)

где - прибыль, включаемая в цену программного средства, руб;

– ставка налога на прибыль;

Таким образом экономический эффект, полученный организацией- разработчиком в виде прироста чистой прибыли от разрабатываемого в дипломном проекте программного продукта составит 1280,8 рублей.

## **6.4 Расчет показателей эффективности инвестиций в разработку программного обеспечения**

Для разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета простой нормы прибыли. Такая рентабельность затрат на разработку программного средства рассчитывается по формуле:

(6.8)

где - прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, р.;

- затраты на разработку программного средства организацией- разработчиком, р.

В соответствии с формулой (6.8), рентабельность инвестиций составит:

Рассчитанный̆ показатель демонстрирует, что разработка имеет положительный экономический эффект.

В процессе технико-экономического обоснования разрабатываемого программного продукта были рассчитаны затраты на его разработку, оценен экономический эффект от его продажи, произведен расчет эффективности инвестиций в разработку и доказана его экономическая целесообразность.

Основываясь на этих расчетах и оценках, можно заключить, что проект является прибыльным, прибыль от продаж составит 1280,8 рублей.

# **ЗАКЛЮЧЕНИЕ**

В процессе разработки дипломного проекта было разработано настольное приложение, предназначенное для настройки и диагностики аппаратных устройств, а именно их отдельных модулей. При разработке как библиотеки взаимодействия программного обеспечения с аппаратным обеспечением, так и при разработке пользовательского интерфейса использовался язык программирования C# и система построения настольных приложений WPF.

Разработанное настольное приложение предоставляет пользователю достигать поставленные в данном дипломном проекте цели, а именно настройка и диагностика отдельных узлов аппаратных устройств.

Приложение разработано с помощью монолитной архитектуры. Пользовательский интерфейс разработан с применением паттерна MVVM, что позволяет легко расширять функциональность приложения без существенного изменения программного кода.

Разработанное в дипломном проекте настольное приложение обладает следующими преимуществами:

* простой и понятный пользователю интерфейс;
* обмен данными с аппаратными устройствами с помощью разработанного протокола обмена данными;
* возможность настройки подключения к аппаратным устройствам;
* возможность отправлять заранее подготовленные команды управления микроконтроллером;
* возможность просматривать полученные ответы от микроконтроллера.

В ходе выполнения данного дипломного проекта были получены новые и структурированы уже имеющиеся знания, полученные за время обучения в университете.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

[1] Настольное приложение [Электронный ресурс]. – Режим доступа: https://freematiq.com/uslugi/desktop-prilozheniya/.

[2] Сравнение монолитной и микросервисной архитектуры [Электронный ресурс]. – Режим доступа: https://www.atlassian.com/ru/microservices/microservices-architecture/microservices-vs-monolith.

[3] MVVM паттерн [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/338518/.

[4] Язык программирования C# [Электронный ресурс]. – Режим доступа: https://timeweb.com/ru/community/articles/chto-takoe-csharp.

[5] WPF [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/WPF/base\_WPF/level1/1\_1.php.

[6] СУБД SQLite [Электронный ресурс]. – Режим доступа: https://unetway.com/tutorial/sqlite.

[7] COM-порт [Электронный ресурс]. – Режим доступа: https://us800.ru/com.htm.

[8] CAN-шина [Электронный ресурс]. – Режим доступа: https://microtronic.com.ua/ru/news/190-can.

# **ПРИЛОЖЕНИЕ А**

**Текст программы**

Конструктор класса CANController

private SerialPort serialPort;

public CANController(string portName, int speedKbit)

{

Console.WriteLine("Creating object of CANControllerClass.....");

var ports = CANUtils.GetAllCOMPorts();

serialPort = new SerialPort(portName, BAUDRATE, Parity.None, 8, StopBits.One);

serialPort.ReadTimeout = READ\_TIMEOUT;

serialPort.WriteTimeout = WRITE\_TIMEOUT;

serialPort.Handshake = Handshake.None;

serialPort.RtsEnable = true;

if (ports.Contains(portName))

{

Connect();

if (!CheckIfCANAdapter())

{

throw new Exception($"Device that connect to {portName} port, is not a CAN adapter!");

}

SetPortSpeed(speedKbit);

}

else

{

throw new Exception($"There is no connected {portName} port!");

}

}

Класс SetupDeviceCommand

public class SetupDeviceCommand : ISetupDeviceCommand

{

public Tuple<int, string, int> GetResetDeviceParameters(string deviceType, int address)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

var resetBytes = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["RESET\_DEVICE\_BYTE"].ToString("X2");

var data = resetBytes + @"FFFFFFFFFFFFFF";

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["PRIORITY"],

ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

public Tuple<int, string, int> GetDisableGainCorrectionParameters(string deviceType, int address)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

var gainCorrectionByte = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["CORRECTION\_BYTE"].ToString("X2");

var data = @"FF" + gainCorrectionByte + @"FFFFFFFFFFFF";

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["PRIORITY"],

ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

public Tuple<int, string, int> GetDisableTemperatureCorrectionParameters(string deviceType, int address)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

var temperatureCorrectionByte = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["CORRECTION\_BYTE"].ToString("X2");

var data = @"FFFF" + temperatureCorrectionByte + @"FFFFFFFFFF";

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["PRIORITY"],

ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

public Tuple<int, string, int> GetLineTest1Parameters(string deviceType, int t1Value, int address)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

var t1ByteValue = t1Value.ToString("X2");

var data = @"FFFFFFFFFFFF" + t1ByteValue + @"FF";

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["PRIORITY"],

ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

public Tuple<int, string, int> GetLineTest2Parameters(string deviceType, int t2Value, int address)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

var t2ByteValue = t2Value.ToString("X2");

var data = @"FFFFFFFFFFFFFF" + t2ByteValue;

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["PRIORITY"],

ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.SETUP\_DEVICE\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

}

Класс RequestParametersCommand

public class RequestParametersCommand : IRequestParametersCommand

{

public Tuple<int, string, int> GetRequestParameters(string deviceType, int address, int commandId, int stateNumber = 0)

{

var code = ChakramProtocolUtils.SetCode(deviceType);

string data = string.Empty;

var commandIdHex = commandId.ToString("X2");

if (commandId == 15)

{

var stateHex = stateNumber.ToString("X2");

data = commandIdHex + stateHex;

}

else

{

data = commandIdHex + @"FF";

}

var id = ChakramProtocolUtils.CollectCommandId(ChakramProtocolConstants.REQUEST\_PARAMETERS\_COMMAND["PRIORITY"],

ChakramProtocolConstants.REQUEST\_PARAMETERS\_COMMAND["COMMAND"],

code,

address);

var dataVolume = ChakramProtocolConstants.REQUEST\_PARAMETERS\_COMMAND["DATA\_VOLUME"];

return Tuple.Create(id, data, dataVolume);

}

}

Класс CANController

public class CANController : CANControllerBase

{

private SerialPort serialPort;

private bool isDisposed = false;

public string Message { get; private set; }

public CANController(string portName, int speedKbit)

{

Console.WriteLine("Creating object of CANControllerClass.....");

var ports = CANUtils.GetAllCOMPorts();

serialPort = new SerialPort(portName, BAUDRATE, Parity.None, 8, StopBits.One);

serialPort.ReadTimeout = READ\_TIMEOUT;

serialPort.WriteTimeout = WRITE\_TIMEOUT;

serialPort.Handshake = Handshake.None;

serialPort.RtsEnable = true;

if (ports.Contains(portName))

{

Connect();

if (!CheckIfCANAdapter())

{

throw new Exception($"Device that connect to {portName} port, is not a CAN adapter!");

}

SetPortSpeed(speedKbit);

}

else

{

throw new Exception($"There is no connected {portName} port!");

}

}

private void Connect()

{

try

{

if (serialPort.IsOpen)

{

Disconnect();

}

serialPort.Open();

Console.WriteLine($"Connected to port {serialPort.PortName} successfully!");

}

catch (Exception)

{

throw;

}

}

private void Disconnect()

{

try

{

serialPort.Close();

Console.WriteLine($"Disconnected from port {serialPort.PortName} successfully");

}

catch (Exception)

{

throw;

}

}

public Queue<byte> ReadDataPacket()

{

var byteQueue = new Queue<byte>();

if (serialPort.IsOpen)

{

while (serialPort.BytesToRead > 0)

{

byte b = (byte)serialPort.ReadByte();

byteQueue.Enqueue(b);

}

}

return byteQueue;

}

public void WriteDataPacket(int id, string packet, int dataVolume)

{

string result = string.Empty;

byte[] bytesId;

try

{

if (id > STANDART\_FRAME)

{

result = "T";

bytesId = Utils.FromIntToByteArray(id);

}

else

{

result = "t";

bytesId = Utils.FromIntToByteArray(id, false);

}

var bytesPacket = Convert.FromHexString(packet);

var prettyPacket = CANUtils.FromByteArrayToPrettyString(bytesPacket);

var prettyBytesId = CANUtils.FromByteArrayToPrettyString(bytesId);

// На данном этапе result будет равен T или t

string prettyResult = $"{result} {prettyBytesId} {dataVolume} {prettyPacket}";

Console.WriteLine($"Result string: {prettyResult}");

Message = prettyResult;

result += Convert.ToHexString(bytesId) + dataVolume.ToString() + packet;

WriteCommand(result);

Console.WriteLine("Data successfully recorded!");

}

catch(Exception)

{

throw;

}

}

Класс ChakramProtocol

public class ChakramProtocol : IDisposable

{

private bool disposed = false;

private CANController can;

private readonly ICoefficientsCorrectionCommand coefficientsCorrectionCommand;

private readonly IRequestParametersCommand requestParametersCommand;

private readonly ISetupDeviceCommand setupDeviceCommand;

private readonly IWorkingParametersCommand workingParametersCommand;

public ChakramProtocol(CANController can,

ICoefficientsCorrectionCommand coefficientsCorrectionCommand,

IRequestParametersCommand requestParametersCommand,

ISetupDeviceCommand setupDeviceCommand,

IWorkingParametersCommand workingParametersCommand)

{

this.can = can;

this.coefficientsCorrectionCommand = coefficientsCorrectionCommand;

this.requestParametersCommand = requestParametersCommand;

this.setupDeviceCommand = setupDeviceCommand;

this.workingParametersCommand = workingParametersCommand;

}

public void ResetDevice(string deviceType, int address)

{

var (id, data, dataVolume) = setupDeviceCommand.GetResetDeviceParameters(deviceType, address);

can.WriteDataPacket(id, data, dataVolume);

}

public void DisableGainCorrection(string deviceType, int address)

{

var (id, data, dataVolume) = setupDeviceCommand.GetDisableGainCorrectionParameters(deviceType, address);

can.WriteDataPacket(id, data, dataVolume);

}

public void DisableTemperatureCorrection(string deviceType, int address)

{

var (id, data, dataVolume) = setupDeviceCommand.GetDisableTemperatureCorrectionParameters(deviceType, address);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetLineTest1(string deviceType, int address, int t1Value)

{

var (id, data, dataVolume) = setupDeviceCommand.GetLineTest1Parameters(deviceType, t1Value, address);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetLineTest2(string deviceType, int address, int t2Value)

{

var (id, data, dataVolume) = setupDeviceCommand.GetLineTest2Parameters(deviceType, t2Value, address);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetPower(string deviceType, int address, int power)

{

var (id, data, dataVolume) = workingParametersCommand.GetPowerParameters(deviceType, address, power);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetFrequency(string deviceType, int address, int freq)

{

var (id, data, dataVolume) = workingParametersCommand.GetFrequencyParameters(deviceType, address, freq);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetPhase(string deviceType, int address, int phase)

{

var (id, data, dataVolume) = workingParametersCommand.GetPhaseParameters(deviceType, address, phase);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetAttenuator(string deviceType, int address, int att)

{

var (id, data, dataVolume) = workingParametersCommand.GetAttenuatorParameters(deviceType, address, att);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetDelay(string deviceType, int address, int delay)

{

var (id, data, dataVolume) = workingParametersCommand.GetDelayParameters(deviceType, address, delay);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetGeneralCoefficient(string deviceType, int address, int generalCoeff)

{

var (id, data, dataVolume) = coefficientsCorrectionCommand.GetGeneralCoefficientParameters(deviceType, address, generalCoeff);

can.WriteDataPacket(id, data, dataVolume);

}

public void SetFrequencyDependentCoefficients(string deviceType, int address, IList<int> coeffs)

{

var (firstId, firstData, firstDataVolume) = coefficientsCorrectionCommand.GetFrequencyDependentFirstPartParameters(deviceType, address, coeffs);

can.WriteDataPacket(firstId, firstData, firstDataVolume);

var (secondId, secondData, secondDataVolume) = coefficientsCorrectionCommand.GetFrequencyDependentSecondPartParameters(deviceType, address, coeffs);

can.WriteDataPacket(secondId, secondData, secondDataVolume);

}

public void RequestParameters(string deviceType, int address, int commandId, int stateNumber = 0)

{

var (id, data, dataVolume) = requestParametersCommand.GetRequestParameters(deviceType, address, commandId, stateNumber);

can.WriteDataPacket(id, data, dataVolume);

}

protected virtual void Dispose(bool disposing)

{

if (disposed)

{

return;

}

if (disposing)

{

// Освобождаем управляемые ресурсы

}

// освобождаем неуправляемые объекты

disposed = true;

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

~ChakramProtocol()

{

Dispose(false);

}

}

Класс DeviceRepository

public class DeviceRepository : IDeviceRepository

{

public async Task<IEnumerable<Device>> GetAllDevicesAsync()

{

using (var connection = new SqliteConnection(DatabaseConfig.ConnectionString))

{

var devices = await connection.QueryAsync<Device>("SELECT Id, Name, Type, Address FROM Device");

return devices;

}

}

public async Task<Device> GetDeviceByIdAsync(int id)

{

using (var connection = new SqliteConnection(DatabaseConfig.ConnectionString))

{

var device = await connection.QueryFirstOrDefaultAsync<Device>("SELECT Id, Name, Type, Address FROM Device WHERE Id = @id",

new { id = id });

return device;

}

}

public async Task<int> CreateDeviceAsync(Device device)

{

using (var connection = new SqliteConnection(DatabaseConfig.ConnectionString))

{

await connection.ExecuteAsync("INSERT INTO Device(Name, Type, Address) VALUES(@Name, @Type, @Address)", device);

return await connection.QueryFirstOrDefaultAsync<int>("SELECT Id FROM Device WHERE Name = @name AND Type = @type AND Address = @address",

new { name = device.Name, type = device.Type, address = device.Address });

}

}

public async Task<int> UpdateDeviceAsync(Device device)

{

using (var connection = new SqliteConnection(DatabaseConfig.ConnectionString))

{

await connection.ExecuteAsync("UPDATE Device SET Name = @Name, Type = @Type, Address = @Address WHERE Id = @Id", device);

return await connection.QueryFirstOrDefaultAsync<int>("SELECT Id FROM Device WHERE Name = @name AND Type = @type AND Address = @address",

new { name = device.Name, type = device.Type, address = device.Address });

}

}

public async Task DeleteDeviceByIdAsync(int id)

{

using (var connection = new SqliteConnection(DatabaseConfig.ConnectionString))

{

await connection.ExecuteAsync("DELETE FROM Device WHERE Id = @id", new { id = id });

}

}

}

XAML код пользовательской части взаимодействия с базой данных

<Label

Content="Devices:"

HorizontalAlignment="Left"

VerticalAlignment="Center"

Grid.Row="1"

Grid.Column="0"

Height="23"

Width="59"/>

<DataGrid x:Name="devicesGrid"

AutoGenerateColumns="False"

ItemsSource="{Binding Devices,

UpdateSourceTrigger=PropertyChanged,

Mode=TwoWay}"

SelectedItem="{Binding SelectedDevice,

Mode=TwoWay}"

Grid.Row="2"

Grid.Column="0"

IsEnabled="{Binding IsOtherEnabled,

UpdateSourceTrigger=PropertyChanged}"

Margin="5, 0, 5, 0" >

<DataGrid.Columns>

<DataGridTextColumn Header="Id"

Width="0.15\*"

x:Name="idColumn"

IsReadOnly="True"

Binding="{Binding Id,

NotifyOnTargetUpdated=True,

UpdateSourceTrigger=PropertyChanged}"

/>

<DataGridTemplateColumn Header="Name"

Width="0.5\*"

x:Name="nameColumn"

>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<TextBlock Text="{Binding Name,

NotifyOnTargetUpdated=True,

UpdateSourceTrigger=PropertyChanged}"

ToolTip="{Binding Name,

NotifyOnTargetUpdated=True,

UpdateSourceTrigger=PropertyChanged}"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTextColumn Header="Type"

Width="0.3\*"

x:Name="typeColumn"

Binding="{Binding Type,

NotifyOnTargetUpdated=True,

UpdateSourceTrigger=PropertyChanged}"

/>

<DataGridTextColumn Header="Address"

Width="0.2\*"

x:Name="addressColumn"

Binding="{Binding Address,

NotifyOnTargetUpdated=True,

UpdateSourceTrigger=PropertyChanged}"

/>

</DataGrid.Columns>

</DataGrid>

<UniformGrid

x:Name="deviceButtonsGrid"

Grid.Column="0"

Grid.Row="3"

Rows="1"

>

<Button

x:Name="collectDataButton"

Content="Get devices"

Command="{Binding CollectDataCommand}"

Height="25"

Grid.Row="3"

Grid.Column="0"

IsEnabled="{Binding IsOtherEnabled,

UpdateSourceTrigger=PropertyChanged}"

Margin="5, 0, 5, 0"/>

<Button

x:Name="addDeviceButton"

Content="Add device"

Command="{Binding AddDeviceCommand}"

Grid.Row="3"

Height="25"

IsEnabled="{Binding IsOtherEnabled,

UpdateSourceTrigger=PropertyChanged}"

Margin="5, 0, 5, 0"/>

<Button

x:Name="updateDeviceButton"

Content="Update device"

Grid.Row="3"

Height="25"

Command="{Binding UpdateDeviceCommand}"

IsEnabled="{Binding IsOtherEnabled,

UpdateSourceTrigger=PropertyChanged}"

Margin="5, 0, 5, 0"/>

<Button

x:Name="deleteDeviceButton"

Content="Delete device"

Height="25"

Command="{Binding DeleteDeviceCommand}"

Grid.Row="3"

Grid.Column="0"

IsEnabled="{Binding IsOtherEnabled,

UpdateSourceTrigger=PropertyChanged}"

Margin="5, 0, 5, 0"/>

</UniformGrid>

Класс MainViewModel

private ObservableCollection<Device> devices = new ObservableCollection<Device>();

public ObservableCollection<Device> Devices

{

get

{

return devices;

}

set

{

if (devices == value)

{

return;

}

devices = value;

OnPropertyChanged(nameof(Devices));

}

}

private async Task ShowData()

{

devices.Clear();

var repository = App.AppHost.Services.GetRequiredService<IDeviceRepository>();

var devicesFromDB = await repository.GetAllDevicesAsync();

if (!devicesFromDB.Any())

{

dialogService.ShowWarningMessage("There is no devices in database yet!");

return;

}

foreach (var dev in devicesFromDB)

{

if (!devices.Any(x => x.Id == dev.Id))

{

devices.Add(dev);

}

}

}

private async Task AddDevice()

{

var addDeviceViewModel = new AddDeviceViewModel(dialogService);

var addDeviceForm = new AddDeviceWindow(addDeviceViewModel);

addDeviceViewModel.OnRequestClose += (s, e) => addDeviceForm.Close();

addDeviceForm.ShowDialog();

var repository = App.AppHost.Services.GetRequiredService<IDeviceRepository>();

var dev = new Device

{

Name = addDeviceViewModel.NameTextBoxValue,

Type = addDeviceViewModel.SelectedType,

Address = Convert.ToInt32(addDeviceViewModel.AddressTextBoxValue)

};

var id = await repository.CreateDeviceAsync(dev);

dev.Id = id;

devices.Add(dev);

}

private async Task UpdateDevice()

{

var repository = App.AppHost.Services.GetRequiredService<IDeviceRepository>();

if (selectedDevice.Id >= 0)

{

await repository.UpdateDeviceAsync(selectedDevice);

}

else

{

dialogService.ShowErrorMessage("Wrong device!");

}

}

private async Task DeleteDevice()

{

var repository = App.AppHost.Services.GetRequiredService<IDeviceRepository>();

if (selectedDevice.Id >= 0)

{

await repository.DeleteDeviceByIdAsync(selectedDevice.Id);

devices.Remove(selectedDevice);

dialogService.ShowMessage("Device delete successfully!");

}

else

{

dialogService.ShowErrorMessage("Wrong device!");

}

}

Класс Device

public class Device

{

public int Id { get; set; } = 1;

public string Name { get; set; }

public string Type { get; set; }

public int Address { get; set; }

}

Класс App

public partial class App : Application

{

public static IHost AppHost { get; private set; }

public App()

{

AppHost = Host.CreateDefaultBuilder()

.ConfigureServices((hostContext, services) =>

{

services.AddScoped<ICoefficientsCorrectionCommand, CoefficientsCorrectionCommand>();

services.AddScoped<IRequestParametersCommand, RequestParametersCommand>();

services.AddScoped<ISetupDeviceCommand, SetupDeviceCommand>();

services.AddScoped<IWorkingParametersCommand, WorkingParametersCommand>();

services.AddScoped<MainViewModel>();

services.AddScoped<ISetupConfig, SetupConfig>();

services.AddScoped<IDeviceRepository, DeviceRepository>();

services.AddScoped<IDialogService, DialogService>();

services.AddSingleton<MainWindow>();

services.AddSingleton<AddDeviceWindow>();

})

.Build();

}

protected override async void OnStartup(StartupEventArgs e)

{

await AppHost.StartAsync();

AppHost.Services.GetRequiredService<ISetupConfig>().Setup();

var startForm = AppHost.Services.GetRequiredService<MainWindow>();

startForm.DataContext = AppHost.Services.GetRequiredService<MainViewModel>();

startForm.Show();

base.OnStartup(e);

}

protected override async void OnExit(ExitEventArgs e)

{

var viewModel = AppHost.Services.GetRequiredService<MainViewModel>();

// Check if "Disconnect" button was pressed before closing window

if (viewModel.IsConnectEnabled == false && viewModel.IsOtherEnabled == true)

{

viewModel.DisconnectFromDevice();

}

await AppHost.StopAsync();

AppHost.Dispose();

base.OnExit(e);

}