МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

# Разработка веб-интерфейса электронного блока управления электрогидравлического усилителя мощности

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.406 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Д.В. Топольский
«___»_____2025 г.

Автор работы,
студентка группы КЭ-406
_____Ж.С. Величко
«___»_____2025 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____ С.В. Сяськов
«___»_____2025 г

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

# ЗАДАНИЕ

**на выпускную квалификационную работу бакалавра**

студентке группы КЭ-406
Величко Жанне Сергеевне
обучающейся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка веб-интерфейса электронного блока управления электрогидравлического усилителя мощности» утверждена приказом по университету от «21» апреля 2025 г. №648-13/12.

2. **Срок сдачи студентом законченной работы:** 1 июня 2025 г.

3. **Исходные данные к работе:** требования к интерфейсу взаимодействия с пользователем.

3.1. Отображение числовых значений (две колонки по три значения) с частотой обновления не менее 10 раз в секунду.

3.2. Возможность настройки отображаемых значений при помощи органов управления (два поворотных энкодера).

3.3. В отдельном окне интерфейса организовать вывод графиков изменения выбранных значений, количество одновременно выводимых графиков не менее 4-х.

3.4. Количество разрядов для отображения числовых значений - не менее 6.

3.5. Язык программирования C/C++.

3.6. Использование графических библиотек для создания пользовательского интерфейса .

4. **Перечень подлежащих разработке вопросов:**

1. Аналитический обзор современной научно-технической, нормативной, методической литературы.
2. Разработка алгоритмов.
3. Разработка и отладка программного кода.
4. Тестирование

5. **Дата выдачи задания:** 2 декабря 2024 г.

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*Ж.С. Величко*/

КАЛЕНДАРНЫЙ ПЛАН

| Этап | Срок сдачи | Подпись руководителя |
|---|---|---|
| Введение и обзор литературы | 03.03.2025 | |
| Разработка алгоритмов | 22.03.2025 | |
| Разработка и отладка программного кода | 12.04.2025 | |
| Тестирование | 26.04.2025 | |
| Компоновка текста работы и сдача на нормоконтроль | 22.05.2025 | |
| Подготовка презентации и доклада | 30.05.2025 | |

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*Ж.С. Величко*/

Аннотация

Ж.С. Величко Разработка веб-интерфейса электронного блока управления электрогидравлического усилителя мощности. - Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 96 с., 8 ил., библиогр. список - 16 наим.

В ходе данной работы был разработан веб-интерфейс электронного блока управления электрогидравлического усилителя мощности на языке C/C++ с использованием библиотеки LVGL. Интерфейс обеспечивает отображение и настройку параметров через TFT-дисплей и два поворотных энкодера, а также визуализацию данных в виде графиков. Реализована эмуляция для ПК с SDL, что облегчает тестирование. Проведённое тестирование подтвердило корректность и соответствие требованиям технического задания.

# ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ ..... 7
1. АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ ..... 10
2. РАЗРАБОТКА АЛГОРИТМОВ ..... 16
2.1. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ..... 16
2.2. СОСТОЯНИЕ ИНТЕРФЕЙСА ..... 17
2.3 АЛГОРИТМ ОБРАБОТКИ ЭНКОДЕРОВ ..... 20
2.4. АЛГОРИТМ ОБНОВЛЕНИЯ ДАННЫХ ..... 21
2.5. АЛГОРИТМ ОТЛАДКИ НА ПК ..... 22
2.6. ВИЗУАЛЬНОЕ ПРЕДСТАВЛЕНИЕ АЛГОРИТМОВ ..... 23
3. РАЗРАБОТКА И ОТЛАДКА ПРОГРАММНОГО КОДА ..... 26
3.1. ОПИСАНИЕ КЛЮЧЕВЫХ МОДУЛЕЙ И АЛГОРИТМОВ ..... 28
3.2. КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС ..... 29
3.3. ОТЛАДКА НА ПК ..... 31
3.4. ВЫВОДЫ ПО ГЛАВЕ ..... 31
4. ТЕСТИРОВАНИЕ ..... 33
4.1. ТЕСТИРОВАНИЕ ГЛАВНОГО ЭКРАНА ..... 33
4.2. ТЕСТИРОВАНИЕ ЭКРАНА РЕДАКТИРОВАНИЯ ПАРАМЕТРА 34
4.3. ТЕСТИРОВАНИЕ ЭКРАНА ГРАФИКОВ ..... 36
3.4. ТЕСТИРОВАНИЕ ОБРАБОТКИ ОШИБОК ..... 38
4.5. ВЫВОДЫ ПО ГЛАВЕ ..... 38
5. ЗАКЛЮЧЕНИЕ ..... 40
БИБЛИОГРАФИЧЕСКИЙ СПИСОК ..... 41
ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОЕКТА ..... 43
ПРИЛОЖЕНИЕ В. ЗАГОЛОВОЧНЫЕ ФАЙЛЫ ..... 88

# ВВЕДЕНИЕ

В современном мире цифровые технологии проникают во все сферы промышленности, предъявляя все более высокие требования к эффективности и удобству использования оборудования. Одним из ключевых аспектов, обеспечивающих взаимодействие человека с техническими системами, является интерфейс взаимодействия с пользователем (UI). Особенно важную роль UI играет в промышленных электронных приборах, где оператору необходимо оперативно отслеживать параметры работы системы, вносить корректировки и контролировать процесс в целом. От качества интерфейса напрямую зависит производительность, безопасность и эргономичность работы.

Существуют различные подходы к реализации интерфейса взаимодействия с пользователем для промышленных систем. Обычно используются готовые решения, такие как панели оператора и дисплейные модули, предлагающие широкий спектр функциональности и возможности программирования.

Панели оператора (Human-Machine-Interface), представляют собой специальные устройства, предназначенные для взаимодействия человека с промышленным оборудованием или автоматизированными системами. Они позволяют оператору видеть состояние оборудования, вносить изменение в параметры работы и контролировать процесс работы.

В отличие от панелей оператора, представляющие собой устройства с собственным процессором, памятью и программным обеспечением, дисплейные модули требуют внешнего контроллера для управления. Их основная задача состоит в визуализации данных, получаемых от внешнего устройства, например, микроконтроллера, датчика, компьютера и т.д.

Как правило, эти решения характеризуются рядом ограничений:

Во-первых, высокая стоимость может стать существенным фактором, особенно при мелкосерийном производстве или в проектах с ограниченным бюджетом. Стоимость самой панели оператора, программного обеспечения и, возможно, обучения персонала может оказаться неподъемной.

Во-вторых, ограниченные возможности кастомизации могут не позволять адаптировать интерфейс к специфическим потребностям конкретного приложения. Часто, даже при наличии возможностей программирования, существует предел в изменении внешнего вида и функциональности, что может быть критично для уникальных или сложных систем.

В-третьих, определенные габариты этих устройств могут быть критичными для встраиваемых систем. Многие промышленные панели оператора разработаны для установки в шкафах управления или на станках, и их размеры могут быть слишком велики для компактных устройств или систем, где пространство ограничено. Кроме того, энергопотребление этих устройств может быть выше, чем у специализированных решений, что также ограничивает их применение в некоторых сценариях.

В рамках данной работы рассматривается задача разработки программного обеспечения для интерфейса взаимодействия с пользователем электронного блока управления электрогидравлического усилителя мощности. В связи с вышеизложенными ограничениями готовых решений, актуальность данной работы обусловлена необходимостью создания компактного, экономичного и гибкого интерфейса, полностью отвечающего специфическим требованиям электронного блока электрогидравлического усилителя мощности.

Целью дипломной работы является разработка программного обеспечения, обеспечивающего эффективное и удобное взаимодействие пользователя с электронным блоком управления электрогидравлического усилителя мощности.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести аналитический обзор научно-технической, нормативной и методической литературы по тематике работы, включая обзор существующих GUI-библиотек для встраиваемых систем.
2. Разработать алгоритмы, реализующие требуемый функционал интерфейса взаимодействия с пользователем, в соответствии с заданными требованиями.
3. Разработать и отладить программный код, реализующий интерфейс взаимодействия с пользователем на выбранном языке программирования (C/C++), с использованием подходящих библиотек.
4. Провести испытания разработанного программного обеспечения на макете электронного блока управления электрогидравлического усилителя мощности для оценки его работоспособности и соответствия требованиям.

Разрабатываемый интерфейс взаимодействия с пользователем будет реализован с учетом следующих технических требований:

1. Отображение числовых значений в виде двух колонок по три значения с частотой обновления не менее 10 раз в секунду.
2. Возможность настройки отображаемых значений при помощи двух поворотных энкодеров.
3. Вывод графиков изменения выбранных значений в отдельном окне интерфейса, с возможностью одновременного отображения не менее 4-х графиков.
4. Количество разрядов для отображения числовых значений - не менее 6.
5. Использование языка программирования C/C++.
6. Использование GUI-библиотек.

# 1. АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ

В данном разделе рассматриваются подходы к созданию графических пользовательских интерфейсов (GUI) для встраиваемых систем, используя готовые GUI-библиотеки. Традиционный подход, предлагающий разработку программного кода для каждого элемента с нуля, заменяется применением предварительно разработанных компонентов, предоставляемых GUI-библиотеками.

Эти библиотеки включают в себя базисные объекты графического интерфейса, такие как кнопки и другие элементы управления. Программист, используя готовый объект, например, "спидометр", избавляется от необходимости его полной разработки, однако при этом он должен написать программный код для специальных атрибутов, таких как позиционирование, цветовая схема, шкала, интервалы градуировки и визуальный стиль. В случае отсутствия необходимых объектов в составе библиотеки, разработчик должен создать их самостоятельно [1].

Для создания графических интерфейсов во встраиваемых устройствах существуют ряд библиотек:

- LVGL;
- uGFX;
- TouchGFX.

В отличие от универсальных GUI-библиотек, библиотеки для встраиваемых систем, согласно [2], характеризуются следующими особенностями:

- оптимизация потребления памяти;
- инкрементальная отрисовка;
- автономность отрисовки;
- аппаратное ускорение;
- гибкость цветовой палитры;
- разнообразие устройств ввода;
- платформенная независимость.

LVGL - это бесплатная графическая библиотека с открытым исходным кодом, предназначенная для создания встроенных GUI, характеризующаяся простотой использования, развитыми визуальными эффектами и малым объемом занимаемой памяти [3].

Согласно [4], LVGL представляет собой библиотеку на языке C (совместимую с C++) без внешних зависимостей, которая может быть скомпилирована для различных микроконтроллеров (MCU) или микропроцессоров (MPU) с любой операционной системой (OS). LVGL поддерживает монохромные, электронные OLED- или TFT-дисплеи, а также мониторы и может использоваться в коммерческих проектах без лицензионных ограничений [5].

Основные функциональные возможности LVGL включают широкий спектр виджетов, расширенные параметры стилизации, поддержку анимации, различные устройства ввода и высокую производительность при низком использовании памяти. Библиотека обладает исчерпывающей документацией, активным сообществом и возможностью получения индивидуальной поддержки [6].

LVGL позволяет разработчикам создавать современные и визуально привлекательные пользовательские интерфейсы, минимизируя объем кода и потребление памяти [7].

Минимальные требования для запуска LVGL:

– 16, 32 или 64-битный микроконтроллер или процессор с тактовой частотой > 16 МГц;
– Flash/ROM > 64 КБ (рекомендуется 180 КБ);
– ОЗУ 8 КБ (рекомендуется 24 КБ);
– компилятор C99 или новее [8].

uGFX представляет собой легковесную библиотеку, предназначенную для дисплеев и сенсорных экранов, предоставляющую функциональность, необходимую для создания полнофункционального встроенного GUI [9]. Библиотека характеризуется небольшим размером и высокой скоростью работы, благодаря отключению неиспользуемых функций.

uGFX - это платформенно независимая, модульная графическая библиотека, написанная на C и предназначенная для взаимодействия с различными типами дисплеев и сенсорными панелями во встраиваемых системах [10].

Основной целью uGFX является предоставление богатого набора возможностей для создания законченного GUI при сохранении минимальных требований к аппаратному обеспечению [11]. Библиотека uGFX полностью написана на языке C, что обеспечивает её совместимость с приложениями на языке C++.

Выделяют следующие достоинства библиотеки uGFX:

– широкие функциональные возможности;
– независимость от аппаратной части и операционной системы;
– поддержка большого количества контроллеров дисплеев и сенсорных панелей;
– открытый исходный код.

К недостаткам можно отнести относительную сложность в освоении и отсутствие русскоязычной документации.

TouchGFX - это графическая программная среда, оптимизированная для микроконтроллеров STM32 [12].

Графическое приложение на базе TouchGFX представляет собой набор "экранов", каждый из которых состоит из графических элементов и соответствующей логики [13].

TouchGFX состоит из трех основных частей - двух инструментов и одного фреймворка [14]:

– TouchGFX Designer (конструктор GUI);
– TouchGFX Generator (плагин STM32CubeMX для настройки и генерации уровня абстракции);
– TouchGFX Engine (фреймворк C++, управляющий приложением пользовательского интерфейса).

TouchGFX применяется в широком спектре приложений, где требуется интуитивно понятный и визуально привлекательный пользовательский интерфейс [15].

Требования к памяти:

- встроенная RAM составляет 10–20 КБ для работы библиотеки и 1–15 КБ для графических элементов;
- встроенная ROM составляет 20 КБ для библиотеки и 1–100 КБ для описаний экранов и логики работы графического интерфейса;
- внешняя RAM зависит от разрешения экрана и количества видеобуферов, например, для дисплея 320×240 QVGA с двумя видеобуферами требуется 307 КБ;
- внешней Flash зависит от количества и размера графических элементов, используемых в приложении, и обычно составляют 1–8 МБ [16].

Выбор GUI-библиотеки для встраиваемой системы является ключевым этапом разработки и требует тщательного анализа, учитывающего специфические требования проекта и возможности каждой библиотеки. Для облегчения процесса выбора, ниже представлена сравнительная таблица основных характеристик библиотек LVGL, uGFX и TouchGFX

Таблица 1 - Сравнительный анализ GUI-библиотек

| **Критерий** | **LVGL** | **uGFX** | **TouchGFX** |
|---|---|---|---|
| **Поддержка MCU/MPU** | Широкая, не зависимая от конкретного производителя | Широкая, платформенно-независимая | Оптимизирована для STM32 |
| **Язык программирования** | C (совместим с C++) | C | C++ |
| **Лицензия** | MIT (бесплатная, открытый исходный код) | BSD (бесплатная, открытый исходный код) | бесплатна для STM32 |
| **Простота использования** | Относительно простая, с обширной документацией и сообществом | Средняя, требует определенной квалификации | Упрощенный дизайн с помощью TouchGFX Designer |

Продолжение таблицы 1

| Критерий | LVGL | uGFX | TouchGFX |
|---|---|---|---|
| **Занимаемая память** | Оптимизированная, низкое потребление RAM/ROM | Оптимизированная, самая низкая среди представленных | Требует больше памяти, чем LVGL и uGFX |
| **Производительность** | Высокая, поддержка аппаратного ускорения | Высокая, но может потребовать оптимизации | Высокая, благодаря оптимизации под STM32 |
| **Удобство разработки** | Развиты API, широкий набор виджетов | Базовый набор виджетов, более низкоуровневый API | Визуальный редактор (TouchGFX Designer), генерация кода |
| **Гибкость и кастомизация** | Высокая, широкие возможности стилизации | Средняя, более ограниченные возможности стилизации | Средняя, ограничения связанные с генерацией кода |
| **Сообщество и поддержка** | Активное сообщество, обширная документация | Меньшее сообщество, документация менее полная | Поддержка от STMicroelectronics |
| **Стоимость** | Бесплатная | Бесплатная | Бесплатна для использования с STM32, необходимо приобретать лицензию при использовании с другими платформами |

# 2. РАЗРАБОТКА АЛГОРИТМОВ

В рамках данной главы представлено описание разработанных алгоритмов, предназначенных для реализации пользовательского интерфейса электронного блока управления электрогидравлического усилителя мощности (ЭГУР). Целью разработки является создание эффективного и интуитивно понятного интерфейса, обеспечивающего оператору возможность мониторинга состояния системы, управления ключевыми параметрами и визуализации данных. В соответствии с техническим заданием, реализация интерфейса осуществляется на базе микроконтроллера с использованием графической библиотеки LVGL и TFT-дисплея, что позволяет создать компактное и экономичное решение.

## 2.1. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Фундаментом программного обеспечения пользовательского интерфейса является модель конечного автомата (КА). Данная модель представляет собой совокупность состояний (экранов), между которыми осуществляется переключение в зависимости от действий пользователя, регистрируемых органами управления (энкодеры и кнопки).

Основной цикл программы реализует следующую последовательность действий:

- регистрация событий (опрос состояния аппаратных органов управления (энкодеров и кнопок) с целью выявления событий (вращение, нажатие));
- обновление данных (получение актуальных значений параметров ЭГУР из соответствующих ячеек памяти микроконтроллера);
- отрисовка интерфейса (динамическое обновление информации, отображаемой на TFT-дисплее, в соответствии с текущим состоянием КА и полученными данными);

– обработка действий пользователя (выполнение действий, соответствующих зарегистрированным событиям от органов управления (нажатие кнопок или поворот энкодера)).

## 2.2. СОСТОЯНИЕ ИНТЕРФЕЙСА

Разработанный пользовательский интерфейс состоит из трех основных состояний (экранов), каждое из которых предоставляет определенный набор функциональных возможностей

Главный экран

Функциональное назначение: отображение основных параметров работы ЭГУР в числовом формате. Данные представлены в виде двух колонок, каждая из которых содержит по три параметра.

Алгоритм работы:

1. Извлечение текущих значений параметров ЭГУР из соответствующих ячеек памяти микроконтроллера (например, давление, температура, ток, угол поворота рулевого колеса).
2. Преобразование числовых значений в строковое представление с фиксированной длиной (не менее 6 символов) для обеспечения единообразного отображения.
3. Вывод отформатированных значений параметров на дисплей в виде двух колонок с использованием виджетов lv_label библиотеки LVGL. Частота обновления отображаемой информации составляет не менее 10 раз в секунду, что обеспечивает визуализацию данных в режиме, близком к реальному времени.
4. Обработка вращения одного из поворотных энкодеров для визуального выделения параметра, выбранного для изменения или отображения на

графике (подробное описание алгоритма обработки энкондеров представленно в разделе 2.3).

Инструкция по использованию:

1. При запуске системы на экране отображаются текущие значения шести параметров ЭГУР.
2. Вращение первого поворотного энкодера позволяет выбрать параметр, значение которого необходимо изменить. Выбранный параметр выделяется визуально (изменение цвета текста, рамка), обеспечивая наглядность выбора.
3. Однократное нажатие на первый поворотный энкодер активирует экран редактирования выбранного параметра.
4. Вращение второго поворотного энкодера позволяет выбрать параметр, график которого необходимо отобразить.
5. Однократное нажатие на второй поворотный энкодер переводит систему в режим отображения графика выбранного параметра.

Экран редактирования параметров системы

Функциональное назначение: Предоставление пользователю возможности изменения значений выбранных параметров ЭГУР.

Алгоритм работы:

1. Вывод на дисплей наименования выбранного параметра и его текущего значения.
2. Обработка вращения поворотного энкодера для увеличения или уменьшения значения параметра с заданным шагом дискретизации.
3. Проверка текущего значения параметра на соответствие допустимому диапазону.
4. Запись измененного значения параметра в соответствующую ячейку памяти микроконтроллера при нажатии на поворотный энкодер.

5. Возврат на главный экран при повторном нажатии на энкодер или по истечении заданного времени бездействия пользователя.

Инструкция по использованию:

1. После выбора параметра на главном экране и нажатия на первый поворотный энкодер, отображается экран редактирования.
2. Вращение поворотного энкодера позволяет изменить значение параметра. Текущее значение отображается на экране.
3. Отображаются минимальное и максимальное допустимые значения параметра.
4. Нажатие на поворотный энкодер сохраняет новое значение параметра и возвращает пользователя на главный экран.

Экран отображения графиков

Функциональное назначение: Визуализация динамики изменения выбранного параметра во времени посредством графического представления.

Алгоритм работы:

1. Создание экземпляра виджета графика (lv_chart) с использованием библиотеки LVGL, задание параметров осей координат, масштаба и цветовой палитры для обеспечения информативного отображения данных.
2. Периодическое считывание значения выбранного параметра и добавление его в массив данных, используемый для построения графика.
3. Обновление виджета графика с использованием массива данных, обеспечивая динамическое отображение изменений параметра.
4. Реализация возможности изменения масштаба графика по осям X (время) и Y (значение параметра) с помощью поворотных энкодеров для детального анализа данных.

5. Реализация возможности выбора других параметров для отображения на графике с использованием энкодеров, обеспечивая гибкость анализа различных характеристик системы.

Инструкция по использованию:

1. После выбора параметра на главном экране и нажатия на второй энкодер, отображается экран графика, демонстрирующий динамику изменения выбранного параметра во времени.
2. На экране отображается график, где по горизонтальной оси представлено время, а по вертикальной - значение параметра.
3. Вращение первого энкодера управляет масштабированием графика по оси времени (X), позволяя увеличивать или уменьшать временной интервал отображения.
4. Вращение второго энкодера управляет масштабированием графика по оси значений параметра (Y), позволяя увеличивать или уменьшать диапазон отображаемых значений.
5. Нажатие на любой из энкодеров инициирует возврат на главный экран.

## 2.3 АЛГОРИТМ ОБРАБОТКИ ЭНКОДЕРОВ

Энкодеры выполняют ключевую роль в навигации по меню, выборе параметров и изменении их значений.

Функция read_encoder(encoder_num) осуществляет чтение состояния указанного энкодера (направление вращения). Возвращаемое значение: 1 (вращение по часовой стрелке), -1 (вращение против часовой стрелки) или 0 (отсутствие вращения).

Алгоритм работы:

1. Считывание состояния: периодическое считывание состояния обоих энкодеров для определения наличия и направления вращения.
2. Определение действия: в зависимости от текущего состояния интерфейса (активного экрана) и номера энкодера, определяются соответствующие действия, которые необходимо выполнить.

Главный экран:

энкодер 1: выбор параметра (перемещение курсора по списку параметров). Переход в режим редактирования (нажатие).

энкодер 2: выбор параметра для отображения на графике. Переход на экран графика (нажатие).

Экран редактирования параметра:

энкодер 1: изменение значения параметра. Сохранение и возврат на главный экран (нажатие).

Экран отображения графиков:

энкодер 1: масштабирование по оси X. Возврат на главный экран (нажатие).

энкодер 2: масштабирование по оси Y. Возврат на главный экран (нажатие).

3. Выполнение действия: выполнение соответствующего действия (изменение значения параметра, переключение между экранами, масштабирование графика).

## 2.4. АЛГОРИТМ ОБНОВЛЕНИЯ ДАННЫХ

Для обеспечения актуальности отображаемой информации, данные на дисплее должны обновляться с частотой не менее 10 Гц.

Функция update_data() осуществляет считывание текущих значений параметров ЭГУР.

Алгоритм:

1. Извлечение значений параметров ЭГУР из соответствующих ячеек памяти или регистров микроконтроллера.
2. Преобразование считанных значений в требуемый формат (например, из целого числа в число с плавающей точкой) для унификации представления данных.
3. Сохранение полученных значений в глобальные переменные, используемые для отрисовки интерфейса, обеспечивая доступность данных для отображения.

Вызов функции update_data(): Функция update_data() вызывается в основном цикле программы с использованием таймера, обеспечивая необходимую частоту обновления информации.

## 2.5. АЛГОРИТМ ОТЛАДКИ НА ПК

В рамках данной работы предусмотрена возможность отладки программного обеспечения на персональном компьютере. Это требует реализации эмуляции работы микроконтроллера и TFT-дисплея.

1. Эмуляция микроконтроллера

Использование кросс-компилятора для сборки кода, предназначенного для микроконтроллера, под архитектуру ПК.

Реализация заглушек (stub-функций) для аппаратных функций, специфичных для микроконтроллера (например, функций работы с портами ввода-вывода, АЦП, таймерами). Эти функции должны имитировать поведение аппаратных устройств, возвращая предопределенные значения или случайные числа в заданном диапазоне для имитации работы датчиков ЭГУР.

2. Эмуляция TFT-дисплея

Использование библиотеки SDL или аналогичной для создания окна на ПК, имитирующего TFT-дисплей.

Перенаправление функций рисования библиотеки LVGL на функции библиотеки SDL для отрисовки графических элементов в окне эмулятора.

3. Интеграция с PlatformIO и Visual Studio Code

Настройка PlatformIO для сборки кода под архитектуру ПК с использованием кросс-компилятора и библиотек эмуляции.

Использование Visual Studio Code для редактирования, сборки и отладки кода.

## 2.6. ВИЗУАЛЬНОЕ ПРЕДСТАВЛЕНИЕ АЛГОРИТМОВ

Для наглядного представления разработанных алгоритмов, на рисунке 1 представлена диаграмма состояний пользовательского интерфейса (графическое представление состояний пользовательского интерфейса и переходов между ними).

Для наглядного представления разработанных алгоритмов, представлены следующие визуальные материалы:

Блок-схема, демонстрирующая логику переключения экранов и обработки пользовательского ввода (рисунок 1).

Диаграмма состояний пользовательского интерфейса (графическое представление состояний пользовательского интерфейса и переходов между ними) (рисунок2).

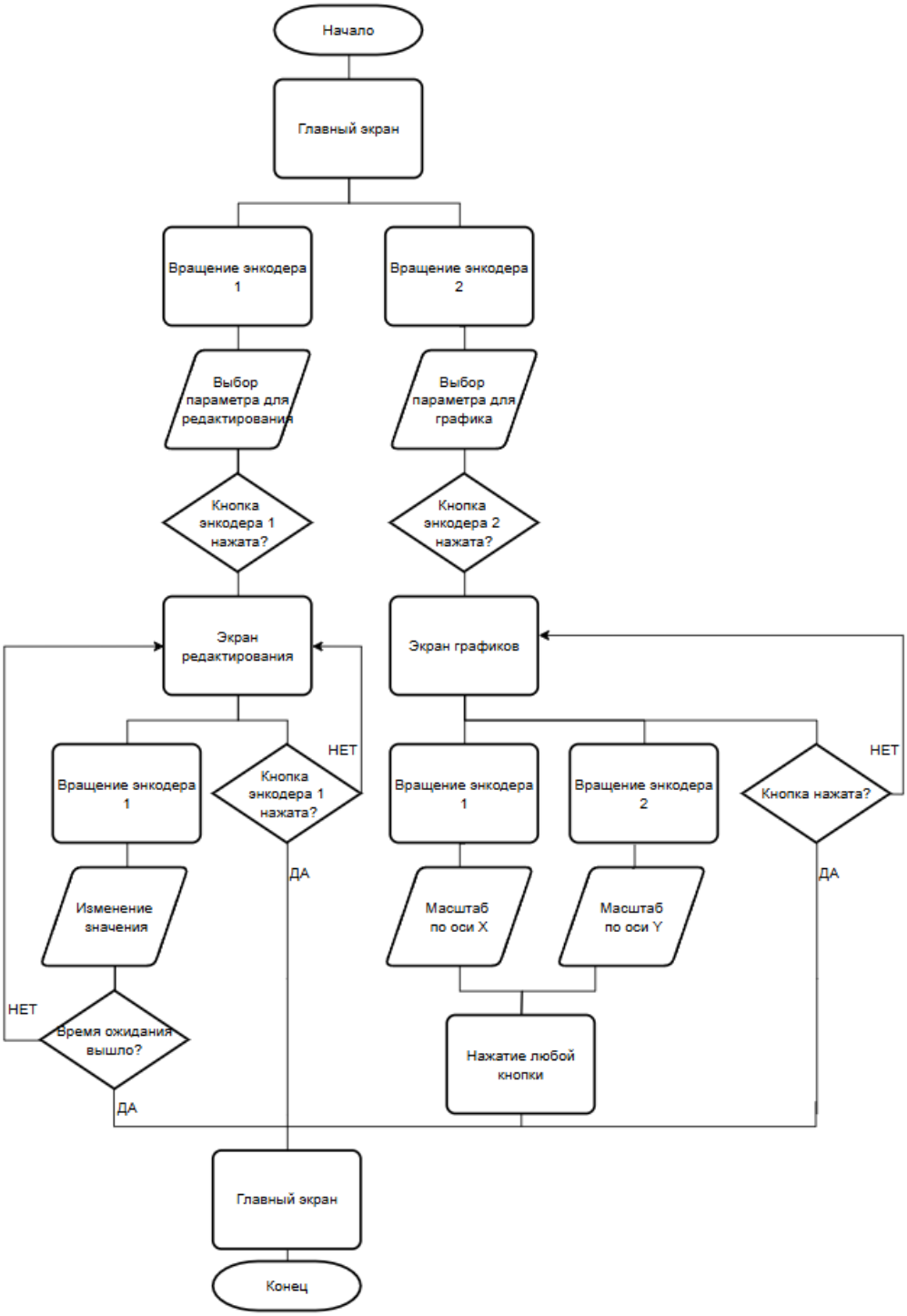


Рисунок 1 - Блок-схема логики переключения экранов

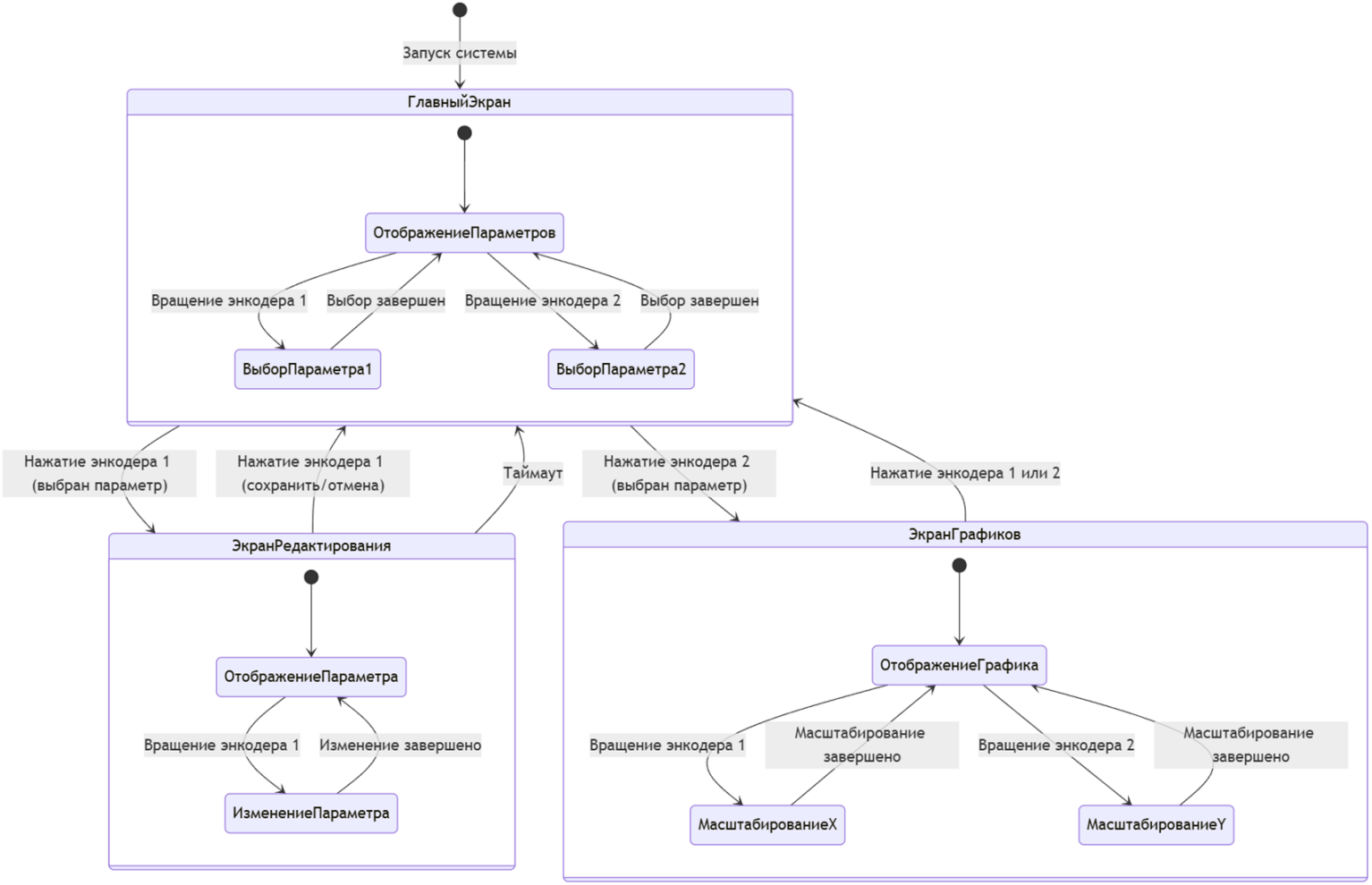


Рисунок 2 - Диаграмма состояний пользовательского интерфейса

# 3. РАЗРАБОТКА И ОТЛАДКА ПРОГРАММНОГО КОДА

В этом разделе необходимо описать методы реализации ПС или ПТС (со схемами, в частности, структурной, функциональной, принципиальной схемой, схемой соединений, подключений, общей, расположения), используемые математические алгоритмы решения задачи (если они есть) и исследовательскую часть, если она необходима для решения проблемы. Вся информация может быть описана как в одном разделе, так и разбита, например, на нижеприведенные подразделы.

В данной главе подробно рассматривается структура программного кода веб-интерфейса ЭГУР, описываются основные модули, их функциональное назначение, принципы взаимодействия, а также детали реализации ключевых алгоритмов.

Проект имеет четкую организацию файлов, разделенную на несколько основных директорий:

1. src/ - Исходный код проекта:
    1.1. main.cpp - Основной файл программы.
    1.2. lvgl_setup.cpp - Настройка библиотеки LVGL.
    1.3. data_manager.cpp - Управление данными параметров.
    1.4. encoders.cpp - Обработка энкодеров.
    1.5. error_handler.cpp - Обработка ошибок.
    1.6. communication.cpp - Коммуникационный интерфейс.
    1.7. ui/ - Файлы пользовательского интерфейса.
        1.7.1. ui.cpp - Общие функции UI.
        1.7.2. screens/ - Экраны интерфейса.
            1.7.2.1. main_screen.cpp - Главный экран.
            1.7.2.2. edit_screen.cpp - Экран редактирования.

1.7.2.3. chart_screen.cpp - Экран графиков.

1.7.3. components/ - Компоненты интерфейса.

1.7.3.1. number_display.cpp - Компонент отображения чисел.

1.8.hal/ - Аппаратно-зависимый слой.

1.8.1. hal_mcu.cpp - Реализация для микроконтроллера.

1.8.2. hal_sdl.cpp - Реализация для SDL (ПК).

2. include/ - Заголовочные файлы:

2.1.config.h - Конфигурация проекта.

2.2.common.h - Общие определения.

2.3.data_manager.h - Интерфейс управления данными.

2.4.encoders.h - Интерфейс работы с энкодерами.

2.5.error_handler.h - Интерфейс обработки ошибок.

2.6.communication.h - Интерфейс коммуникации.

2.7.lvgl_setup.h - Настройка LVGL.

2.8.ui/ - Заголовочные файлы UI.

2.8.1. ui.h - Общие определения UI.

2.8.2. screens/ - Заголовки экранов.

2.8.3. components/ - Заголовки компонентов.

2.9.hal/ - Заголовки HAL.

3. lib/ - Библиотеки.
4. Proteus/ - Файлы проекта Proteus для симуляции.
5. .pio/ - Файлы PlatformIO.
6. .vscode/ - Настройки Visual Studio Code.

## 3.1. ОПИСАНИЕ КЛЮЧЕВЫХ МОДУЛЕЙ И АЛГОРИТМОВ

Программное обеспечение веб-интерфейса ЭГУР состоит из нескольких ключевых компонентов, каждый из которых отвечает за определенную функциональность.

Основной модуль (main.cpp) является точкой входа в программу. Он выполняет инициализацию аппаратного обеспечения, настройку библиотеки LVGL, инициализацию конечного автомата и запускает основной цикл программы (Листинг А.1).

Основной цикл программы включает в себя: регистрацию событий ввода пользователя, обновление данных, отрисовку пользовательского интерфейса и обработку действий пользователя. Модуль main.cpp также отвечает за переключение между состояниями (экранами) пользовательского интерфейса в соответствии с моделью конечного автомата.

Модуль управления данными (data_manager.cpp) отвечает за хранение, обновление и предоставление доступа к параметрам ЭГУР (Листинг А.3).

Он содержит массивы для хранения текущих значений параметров (param_values[]) и истории их изменений (param_history[][]). Предоставляет функции для получения и установки значений параметров, их минимальных и максимальных границ, единиц измерения и истории изменений. Функция update_egu_data() обновляет значения параметров либо путем чтения данных с аналоговых входов (в режиме микроконтроллера), либо путем генерации случайных флуктуаций (в режиме симуляции на ПК).

Модули пользовательского интерфейса (ui/, screens/, components/) реализуют отображение информации на TFT-дисплее и взаимодействие с пользователем.

Реализация пользовательского интерфейса:

– main_screen.cpp: реализует главный экран интерфейса, отображающий список параметров и их текущие значения. Обеспечивает навигацию по списку параметров с помощью энкодера и переход к экранам редактирования и графиков (Листинг А.8);
– edit_screen.cpp: реализует экран редактирования выбранного параметра, позволяющий пользователю изменять его значение с помощью энкодера. Обеспечивает сохранение изменений и возврат на главный экран (Листинг А.9);
– chart_screen.cpp: реализует экран отображения графика изменения выбранного параметра во времени. Обеспечивает масштабирование графика по осям X и Y с помощью энкодеров (Листинг А.10).

Модуль обработки энкодеров (encoders.cpp) инициализирует энкодеры, считывает их состояние (направление вращения и нажатие кнопки) и предоставляет эту информацию другим модулям для управления интерфейсом (Листинг А.4).

Аппаратно-зависимый слой (HAL) (hal/) абстрагирует доступ к аппаратным ресурсам микроконтроллера и обеспечивает возможность работы проекта как на реальном оборудовании (с использованием hal_mcu.cpp (Листинг А.12)), так и в режиме эмуляции на ПК (с использованием hal_sdl.cpp (Листинг А.13)). Это позволяет упростить процесс разработки и отладки, а также обеспечить переносимость кода.

## 3.2. КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС

Коммуникационный интерфейс реализован в файлах communication.h и communication.cpp. Он обеспечивает обмен данными между интерфейсом ЭГУР и внешними системами через последовательный порт (Листинг А.6).

Протокол связи имеет следующий формат:

– начальный байт: 0xAA;
– команда: 1 байт;
– данные: переменная длина в зависимости от команды;
– конечный байт: 0xFF.

Функция comm_process() периодически проверяет наличие входящих данных и обрабатывает их. Полученные команды парсятся в функции comm_parse_command() и выполняются соответствующие действия.

Протокол поддерживает следующие команды, определенные в communication.h:

– CMD_GET_PARAM (0x01) - получение значения параметра;
– CMD_SET_PARAM (0x02) - установка значения параметра;
– CMD_GET_ALL_PARAMS (0x03) - получение всех параметров;
– CMD_GET_STATUS (0x04) - получение статуса системы;
– CMD_GET_HISTORY (0x05) - получение истории изменения параметра;
– CMD_GET_ERRORS (0x06) - получение списка ошибок;
– CMD_RUN_DIAG (0x07) - запуск диагностики;
– CMD_EXIT_DIAG (0x08) - выход из режима диагностики.

Для отправки данных используются следующие функции:

– comm_send_param() - отправка значения параметра;
– comm_send_all_params() - отправка всех параметров;
– comm_send_history() - отправка истории изменения параметра.

В режиме отладки на ПК коммуникация осуществляется через виртуальный COM-порт, а в режиме работы на микроконтроллере - через аппаратный UART.

## 3.3. ОТЛАДКА НА ПК

Для отладки на ПК реализована эмуляция работы микроконтроллера. Это позволяет разрабатывать и тестировать интерфейс без использования реального оборудования.

Эмуляция включает:

1. Использование кросс-компилятора для сборки кода под архитектуру ПК.
2. Реализацию заглушек для аппаратных функций.
3. Генерацию симулированных данных для параметров ЭГУР.

В режиме отладки на ПК используется условная компиляция с макросом __PC__, который определяет, какие части кода должны выполняться.

Для эмуляции TFT-дисплея на ПК используется библиотека SDL. Она создает окно, имитирующее дисплей, и перенаправляет функции рисования библиотеки LVGL на функции SDL.

Настройка эмуляции дисплея осуществляется в файле lvgl_setup.cpp (Листинг А.2), который содержит две реализации: одну для микроконтроллера и одну для ПК с использованием SDL.

В режиме отладки на ПК управление осуществляется с помощью клавиатуры, эмулирующей работу энкодеров:

– энкодер 1: A (против часовой), D (по часовой), S (кнопка);
– энкодер 2: J (против часовой), L (по часовой), K (кнопка);
– ESC: Выход из приложения.

## 3.4. ВЫВОДЫ ПО ГЛАВЕ

В данной главе представлен подробный обзор структуры программного кода веб-интерфейса ЭГУР. Описаны основные модули, их функциональное назначение, принципы взаимодействия, а также детали реализации ключевых

алгоритмов, таких как обработка пользовательского ввода, логика переключения между экранами и коммуникационный протокол. Особое внимание уделено организации модульной архитектуры, обеспечивающей удобство сопровождения и масштабирования проекта.

В режиме эмуляции на ПК реализованы заглушки аппаратных функций и эмуляция дисплея с использованием библиотеки SDL, что позволяет полноценно разрабатывать и отлаживать программное обеспечение без необходимости подключения реального оборудования. Такой подход значительно ускоряет процесс тестирования и выявления ошибок, обеспечивая корректность работы интерфейса и его функциональных компонентов до переноса на целевую платформу.

# 4. ТЕСТИРОВАНИЕ

В данной главе описывается методика тестирования разработанного программного обеспечения пользовательского интерфейса электронного блока управления электрогидравлического усилителя мощности (ЭГУР), а также представлены результаты моделирования работы системы и визуализации интерфейса. Поскольку проведение физического тестирования на реальном оборудовании не представилось возможным, основное внимание уделено описанию подходов к тестированию и демонстрации ожидаемых результатов работы системы на основе сгенерированных данных и эмуляции.

Для проведения тестирования использовался режим компиляции проекта под платформу ПК (определяемый макросом __PC__) с использованием SDL для эмуляции TFT-дисплея.

Эмуляция пользовательского ввода осуществлялась с помощью клавиатуры, имитирующей вращение и нажатия энкодеров.

Отладочная информация выводилась в консоль для отслеживания состояния программы и действий пользователя.

Для визуализации интерфейса использовалось окно SDL с разрешением 320×240 пикселей, соответствующим параметрам TFT-дисплея.

Для формирования изображений экранов использовались штатные возможности LVGL и SDL, дополнительно сгенерированы скриншоты с помощью эмулятора.

## 4.1. ТЕСТИРОВАНИЕ ГЛАВНОГО ЭКРАНА

При запуске приложения отображается главный экран (рисунок 3) с двумя колонками параметров, каждая из которых содержит по три параметра с текущими значениями и единицами измерения.

Значения параметров обновляся с частотой не менее 10 Гц, что обеспечивает плавное и своевременное отображение динамики.

Также была протестирована возможность выбора параметра с помощью вращения энкодера 1, выделенный параметр подсвечивается (изменяется цвет фона и цвет текста). Однократное нажатие кнопки энкодера 1 вызывает переход на экран редактирование выбранного параметра.

Вращение энкодера 2 позволяет выбрать параметр для отображения на графике. Нажатие кнопки энкодера 2 вызывает переход на экран графиков.

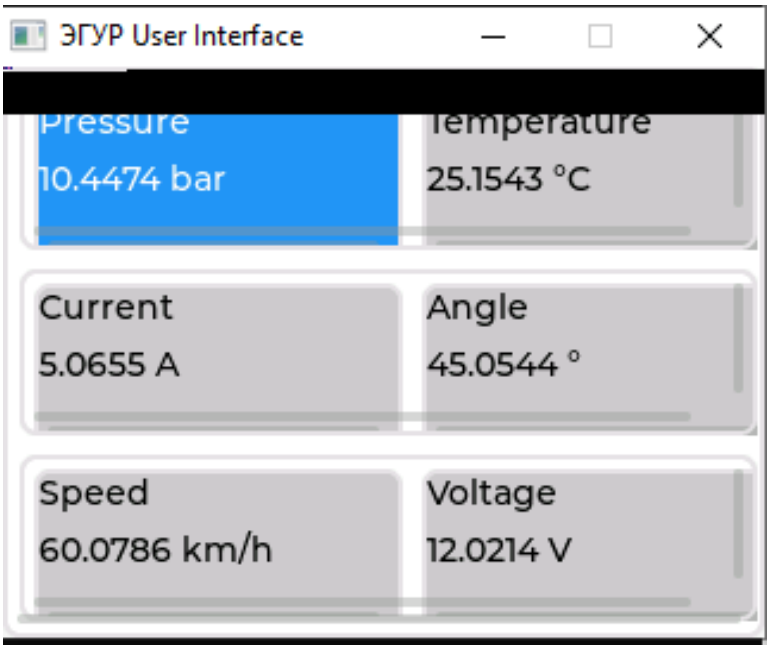


Рисунок 3 - главный экран интерфейса

## 4.2. ТЕСТИРОВАНИЕ ЭКРАНА РЕДАКТИРОВАНИЯ ПАРАМЕТРА

После перехода на экран редактирования (рисунок 4) отображается название параметра, текущее значение и слайдер для визуального изменения.

Вращение энкодера 1 изменяет значение параметра с шагом, соответствующим диапазону (например, для температуры — 1.4 °C). Значение параметра ограничивается минимальными и максимальными пределами, установленными в конфигурации.

Нажатие кнопки энкодера 1 сохраняет измененное значение и возвращает на главный экран. Нажатие кнопки энкодера 2 отменяет изменения и также возвращает на главный экран.

В консоль выводится подробная информация о текущем значении и действиях пользователя (рисунок 5).

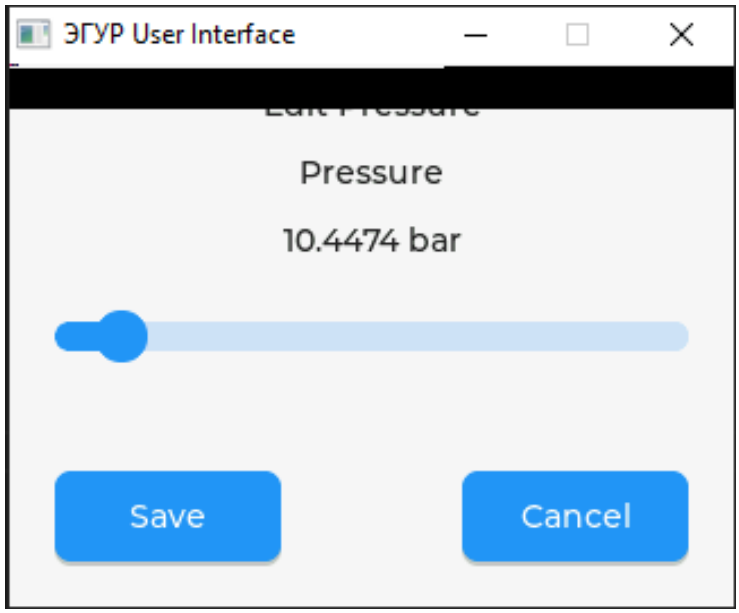


Рисунок 4 - экран редактирования параметров

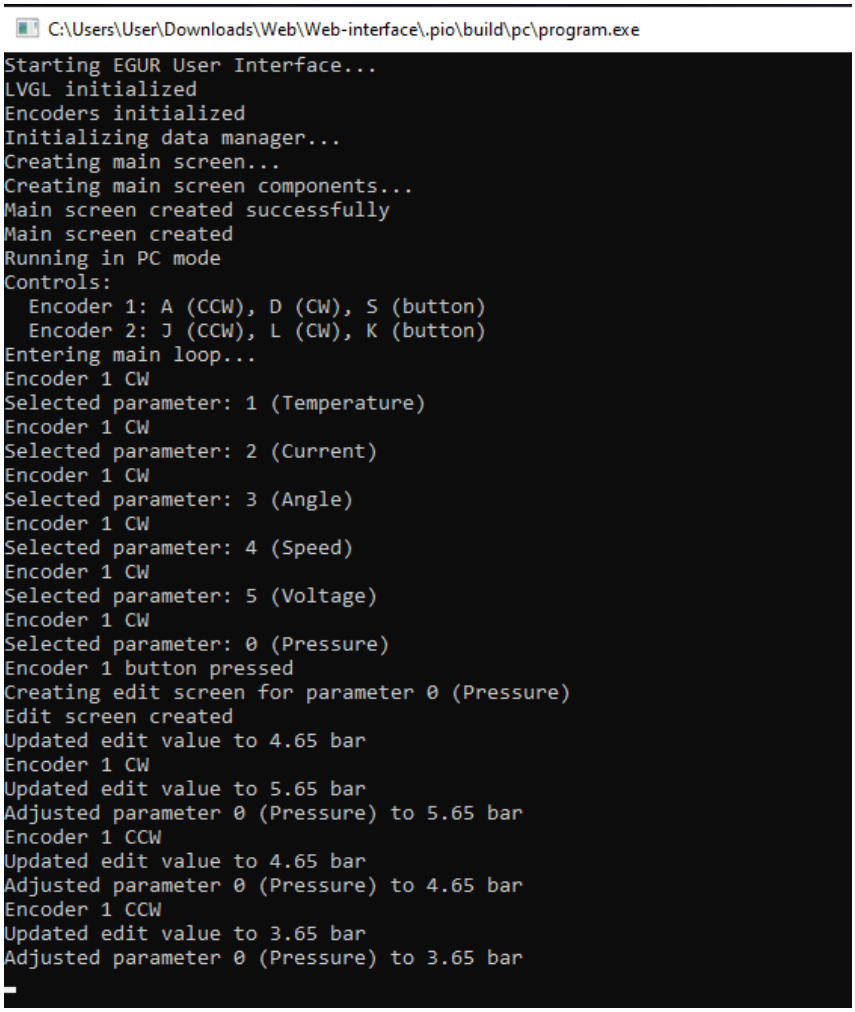


Рисунок 5 - отображение изменения значения параметра в консоли

### 4.3. ТЕСТИРОВАНИЕ ЭКРАНА ГРАФИКОВ

При переходе на экран графиков (рисунок 6) отображается 4 линейных графика выбранных параметров. Данные обновляются динамически, отражая изменения параметра.

Вращение энкодера 1 изменяет масштаб по оси X (число отображаемых точек), позволяя сузить или расширить временной интервал.

Вращение энкодера 2 изменяет масштаб по оси Y (диапазон значений), позволяя увеличить или уменьшить вертикальное масштабирование.

Любое нажатие кнопки возвращает пользователя на главный экран.

В консоль выводятся сообщения о текущих изменениях масштаба (рисунок 7).

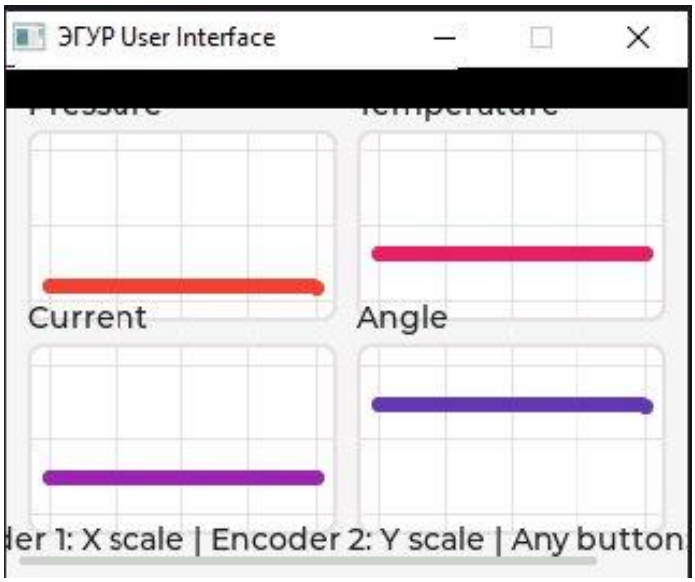


Рисунок 6 - Экран графиков

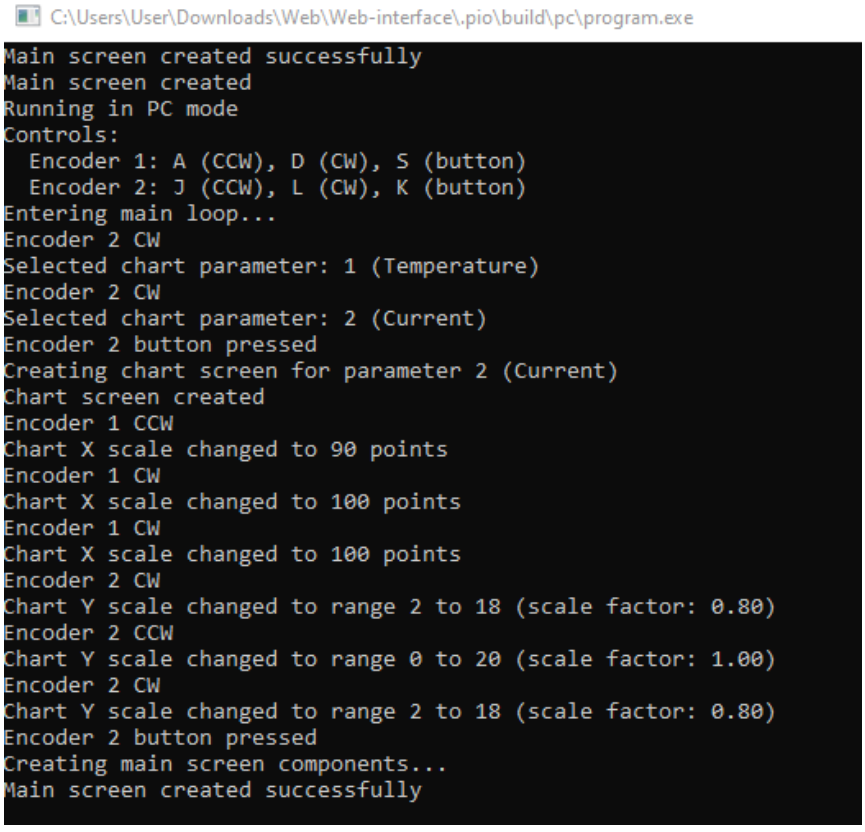


Рисунок 7 - отображение изменения масштаба графика в консоли

### 3.4. ТЕСТИРОВАНИЕ ОБРАБОТКИ ОШИБОК

Для проверки отображения ошибок была выполнена имитация активации нескольких ошибок (например, ошибка датчика давления и температуры).

Отображался экран ошибок с заголовком красного цвета и списком активных ошибок (рисунок 8).

Внизу экрана выводились инструкции по возврату в главное меню.

Имитация экрана ошибок позволила проверить визуальное оформление и корректность работы компонентов.


Рисунок 8 - экран ошибок

### 4.5. ВЫВОДЫ ПО ГЛАВЕ

Разработанное программное обеспечение успешно работает в режиме эмуляции на ПК, полностью обеспечивая функциональность, заявленную в техническом задании.

Эмуляция работы энкодеров с помощью клавиатуры позволяет полноценно тестировать логику интерфейса без необходимости подключения реального оборудования.

Интерфейс корректно отображает данные, реагирует на пользовательский ввод, обеспечивает навигацию между экранами и управление параметрами.

Механизм таймаута бездействия работает стабильно, гарантируя возврат к главному экрану.

Обработка ошибок и их отображение реализованы и визуализируются корректно.

Визуальные элементы интерфейса (списки, слайдеры, графики) отображаются адекватно и обновляются своевременно.

Рекомендации для дальнейших тестирований:

- провести интеграционное тестирование на реальном оборудовании с TFT-дисплеем и энкодерами для проверки взаимодействия с физическими устройствами;
- реализовать автоматизированные тесты для модулей обработки данных и коммуникации;
- расширить тестирование сценариев ошибок с подключением реальных датчиков;
- добавить тестирование производительности и устойчивости при длительной работе.

Таким образом, несмотря на отсутствие возможности физического тестирования на реальном оборудовании, полноценное тестирование в режиме симуляции подтвердило корректность работы разработанного программного обеспечения пользовательского интерфейса ЭГУР и соответствие его функционала требованиям технического задания.

# 5. ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы была достигнута цель - разработан веб-интерфейс электронного блока управления электрогидравлического усилителя мощности.

В рамках работы были выполнены следующие задачи:

1. Проведен аналитический обзор научно-технической, нормативной и методической литературы.
2. Разработаны алгоритмы.
3. На основе разработанных алгоритмов реализован программный код веб-интерфейса ЭГУР.
4. Проведено тестирование.

Проведённое тестирование в режиме симуляции подтвердило корректность работы разработанного программного обеспечения пользовательского интерфейса ЭГУР и соответствие его функционала требованиям технического задания.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. GUI во встраиваемых системах [Электронный ресурс] - URL: https://www.itweek.ru/industrial/article/detail.php?ID=72569 (дата обращения: 25.02.2025).
2. Ключарёв, А. А. Программирование микроконтроллеров STM32: учебное пособие / А. А. Ключарёв, К. А. Кочин, А. А. Фоменкова. - СПб.: ГУАП, 2023. - С. 118.
3. Introduction. [Электронный ресурс] - URL: https://docs.lvgl.io/master/intro/introduction.html (дата обращения: 18.12.2024).
4. LVGL [Электронный ресурс] - URL: https://github.com/lvgl/lvgl (дата обращения: 18.12.2024).
5. Light and Versatile Graphics Library [Электронный ресурс] – URL: https://lvgl.io/ (дата обращения: 28.12.2024).
6. Light and Versatile Graphics Library [Электронный ресурс] - URL: https://www.st.com/en/partner-products-and-services/lvgl.html (дата обращения: 20.02.2025).
7. Что такое LVGL и как она работает во встраиваемых системах [Электронный ресурс] – URL: https://www.rlocman.ru/review/article.html?di=672011 (дата обращения: 20.02.2025).
8. LVGL Graphics Library [Электронный ресурс] - URL: https://docs.espressif.com/projects/esp-iot-solution/en/latest/gui/lvgl_gui.html (дата обращения: 21.02.2025).
9. What is uGFX? [Электронный ресурс] - URL: https://ugfx.io/overview (дата обращения: 05.01.2025).

10. uGFX - библиотека GUI для микроконтроллеров [Электронный ресурс] - URL: https://mazko.github.io/blog/posts/2017/05/20/ugfx-biblioteka-gui-dlia-mikrokontrollerov/ (дата обращения: 20.02.2025).
11. Курниц, А.А. uGFX — графическая библиотека для микроконтроллеров/ А.А. Курниц // КОМПОНЕНТЫ И ТЕХНОЛОГИИ. - 2014. - №10 (159). - С. 97-104.
12. TouchGFXDesigne [Электронный ресурс] - URL: https://www.st.com/en/development-tools/touchgfxdesigner.html (дата обращения: 05.02.2025).
13. Долгушин, С.Ф. Графическая библиотека TouchGFX/ С.Ф. Долгушин // КОМПОНЕНТЫ И ТЕХНОЛОГИИ. - 2018. - №6 (203). - С. 90-96.
14. What is TouchGFX? [Электронный ресурс] - URL: https://support.touchgfx.com/docs/introduction/what-is-touchgfx (дата обращения: 10.02.2025).
15. Getting Started with TouchGFX: Creating Your First UI [Электронный ресурс] - URL: https://blog.embeddedexpert.io/?p=2803 (дата обра-щения: 21.02.2025).
16. Графическая библиотека TouchGFX [Электронный ресурс] - URL: https://kit-e.ru/touchgfx/ (дата обращения: 23.02.2025).

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОЕКТА

Листинг А.1 - Основной файл программы

```
#include <iostream>
#include <thread>
#include <chrono>
#include "lvgl.h"
#include "lvgl_setup.h"
#include "ui/ui.h"
#include "ui/screens/main_screen.h"
#include "ui/screens/edit_screen.h"
#include "ui/screens/chart_screen.h"
#include "encoders.h"
#include "data_manager.h"
#include "config.h"
#include "common.h"
#include "communication.h"
#include "localization.h"
#include "error_handler.h"
#include "diagnostics.h"

#ifdef __PC__
#include <C:\msys64\mingw64\include\SDL2\SDL.h>
#else
#include <Arduino.h>
#endif

#ifdef __PROTEUS__
#include "proteus_interface.h"
#endif

extern int edit_param_id;
extern float edit_value;

// Добавляем обработку таймаута бездействия
#define INACTIVITY_TIMEOUT 30000  // 30 секунд
```

Продолжение листинга А.1

```
// Временная метка последней активности
unsigned long last_activity_time = 0;

// Текущий экран - определяем здесь как основной
ScreenType current_screen = SCREEN_MAIN;

// Обновление временной метки активности
void update_activity_timestamp() {
    last_activity_time = millis();
}

// Проверка таймаута бездействия
void check_inactivity_timeout() {
    if (current_screen != SCREEN_MAIN) {
        if (millis() - last_activity_time > INACTIVITY_TIMEOUT) {
            // Возврат на главный экран после таймаута
            current_screen = SCREEN_MAIN;
            lv_obj_clean(lv_scr_act());
            create_main_screen();
             update_activity_timestamp(); // Сброс временной метки после возврата
на главный экран
        }
    }
}
// Объявляем энкодеры
Encoder encoder1;
Encoder encoder2;

int main(int argc, char* argv[]) {
    std::cout << "Запуск пользовательского интерфейса EGUR..." << std::endl;

#ifdef __PROTEUS__
    // Инициализация интерфейса Proteus
    proteus_init();
#endif
```

Продолжение листинга А.1

```
    // Инициализация LVGL
    lvgl_init();
    std::cout << "LVGL инициализирован" << std::endl;
    // Инициализация энкодеров
    encoder_init(&encoder1, ENCODER_1_PIN_A, ENCODER_1_PIN_B, ENCODER_1_SW_PIN);
    encoder_init(&encoder2, ENCODER_2_PIN_A, ENCODER_2_PIN_B, ENCODER_2_SW_PIN);
    std::cout << "Энкодеры инициализированы" << std::endl;
    // Инициализация менеджера данных
    std::cout << "Инициализация менеджера данных..." << std::endl;
    // Здесь можно добавить инициализацию менеджера данных при необходимости

    // Создание главного экрана
    std::cout << "Создание главного экрана..." << std::endl;
    create_main_screen();
    std::cout << "Главный экран создан" << std::endl;

    update_activity_timestamp(); // Инициализация временной метки активности

#ifdef __PC__
    std::cout << "Запуск в режиме ПК" << std::endl;
    std::cout << "Управление:" << std::endl;
    std::cout << "  Энкодер 1: A (против часовой), D (по часовой), S (кнопка)" <<
std::endl;
    std::cout << "  Энкодер 2: J (против часовой), L (по часовой), K (кнопка)" <<
std::endl;

    // Главный цикл для ПК
    std::cout << "Вход в главный цикл..." << std::endl;
    while (true) {
        // Обработка событий SDL
        SDL_Event event;
        while (SDL_PollEvent(&event)) {
            if (event.type == SDL_QUIT) {
                return 0;
            }
```

Продолжение листинга А.1

```
            // Обработка нажатий клавиш для имитации энкодеров
            if (event.type == SDL_KEYDOWN) {
                  update_activity_timestamp(); // Обновляем временную метку при
нажатии клавиши

                switch (event.key.keysym.sym) {
                    case SDLK_a: // Энкодер 1 против часовой стрелки
                        encoder1.direction = -1;
                        std::cout << "Энкодер 1 против часовой" << std::endl;
                        break;
                    case SDLK_d: // Энкодер 1 по часовой стрелке
                        encoder1.direction = 1;
                        std::cout << "Энкодер 1 по часовой" << std::endl;
                        break;
                    case SDLK_s: // Кнопка энкодера 1
                        encoder1.button_pressed = true;
                        std::cout << "Кнопка энкодера 1 нажата" << std::endl;
                        break;
                    case SDLK_j: // Энкодер 2 против часовой стрелки
                        encoder2.direction = -1;
                        std::cout << "Энкодер 2 против часовой" << std::endl;
                        break;
                    case SDLK_l: // Энкодер 2 по часовой стрелке
                        encoder2.direction = 1;
                        std::cout << "Энкодер 2 по часовой" << std::endl;
                        break;
                    case SDLK_k: // Кнопка энкодера 2
                        encoder2.button_pressed = true;
                        std::cout << "Кнопка энкодера 2 нажата" << std::endl;
                        break;
                    case SDLK_ESCAPE: // Выход
                        return 0;
                }
            }
        }
```

Продолжение листинга А.1

```
        // Обновление данных
        update_egu_data();

        // Обработка обновлений UI на основе состояния энкодеров
        switch (current_screen) {
            case SCREEN_MAIN:
                // Обновление главного экрана
                update_main_screen();

                // Обработка энкодера 1 (выбор параметра)
                if (encoder1.direction != 0) {
                    select_parameter(encoder1.direction);
                    update_activity_timestamp(); // Обновляем временную метку при
вводе энкодера
                }

                // Обработка кнопки энкодера 1 (редактирование параметра)
                if (encoder1.button_pressed) {
                    current_screen = SCREEN_EDIT;
                    lv_obj_clean(lv_scr_act());
                    create_edit_screen_content(get_selected_parameter());
                    update_activity_timestamp(); // Обновляем временную метку при
смене экрана
                }

                // Обработка кнопки энкодера 2 (показ графика)
                if (encoder2.button_pressed) {
                    current_screen = SCREEN_CHART;
                    lv_obj_clean(lv_scr_act());
                     create_chart_screen(get_selected_parameter()); // Используем
выбранный параметр для графика
                    update_activity_timestamp(); // Обновляем временную метку при
смене экрана
                }
                break;
```

Продолжение листинга А.1

```
            case SCREEN_EDIT:
                update_edit_screen();

                if (encoder1.direction != 0) {
                    adjust_parameter_value(encoder1.direction);
                    update_activity_timestamp();
                }
                if (encoder1.button_pressed) {
                    // Сохраняем значение
                    save_parameter_value(edit_param_id, edit_value);

                    // Возврат на главный экран
                    current_screen = SCREEN_MAIN;
                    lv_obj_clean(lv_scr_act());
                    create_main_screen();
                    update_activity_timestamp();
                }
                break;

            case SCREEN_CHART:
                // Обновление графика
                update_chart_screen();
                // Обработка энкодера 1 (масштаб по X)
                if (encoder1.direction != 0) {
                    chart_scale_x(encoder1.direction);
                    update_activity_timestamp(); // Обновляем временную метку при
вводе энкодера
                }

                // Обработка энкодера 2 (масштаб по Y)
                if (encoder2.direction != 0) {
                    chart_scale_y(encoder2.direction);
                    update_activity_timestamp(); // Обновляем временную метку при
вводе энкодера
                }
```

Продолжение листинга А.1

```
                // Обработка любой кнопки (возврат на главный экран)
                if (encoder1.button_pressed || encoder2.button_pressed) {
                    current_screen = SCREEN_MAIN;
                    lv_obj_clean(lv_scr_act());
                    create_main_screen();
                    update_activity_timestamp(); // Обновляем временную метку при
смене экрана
                }
                break;
        }

        // Обработка задач LVGL
        lv_timer_handler();

        // Проверка таймаута бездействия
        check_inactivity_timeout();

        // Сброс состояний энкодеров
        encoder1.button_pressed = false;
        encoder1.direction = 0;
        encoder2.button_pressed = false;
        encoder2.direction = 0;
        // Небольшая задержка
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }

#else
    // Для микроконтроллера
    while (true) {
        // Считываем энкодеры
        encoder_read(&encoder1);
        encoder_read(&encoder2);

        // Обновление данных
        update_egu_data();
```

Окончание листинга А.1

```
        // Обработка обновлений UI на основе состояния энкодеров
        // (Такая же логика, как в режиме ПК)

        // Обработка задач LVGL
        lv_timer_handler();

        // Небольшая задержка
        delay(10);
    }
#endif

return 0;
}
```

Листинг А.2 - Настройка библиотеки LVGL

```
#include "lvgl_setup.h"
#include "lvgl.h"
#include "config.h"
#include <cstdio>  // Для printf

#ifdef __PC__
// Для ПК с SDL
#include <SDL2/SDL.h>

// SDL окно и рендерер
static SDL_Window* window;
static SDL_Renderer* renderer;
static SDL_Texture* texture;

// Буфер дисплея
static lv_disp_draw_buf_t disp_buf;
static lv_color_t buf[SCREEN_WIDTH * 10];

// Драйвер дисплея LVGL
static lv_disp_drv_t disp_drv;
```

Продолжение листинга А.2

```
// SDL callback для сброса буфера LVGL
static void sdl_flush_cb(lv_disp_drv_t* disp_drv, const lv_area_t* area,
lv_color_t* color_p) {
    // Создаем буфер для пиксельных данных
    uint32_t* pixels = new uint32_t[SCREEN_WIDTH * SCREEN_HEIGHT];

    // Конвертируем формат цвета LVGL в формат SDL
    for (int y = area->y1; y <= area->y2; y++) {
        for (int x = area->x1; x <= area->x2; x++) {
            int idx = y * SCREEN_WIDTH + x;
            if (idx < SCREEN_WIDTH * SCREEN_HEIGHT) {
                pixels[idx] = lv_color_to32(*color_p);
                color_p++;
            }
        }
    }

    // Обновляем текстуру новыми пиксельными данными
    SDL_UpdateTexture(texture, NULL, pixels, SCREEN_WIDTH * sizeof(uint32_t));

    // Рендерим текстуру на экран
    SDL_RenderClear(renderer);
    SDL_RenderCopy(renderer, texture, NULL, NULL);
    SDL_RenderPresent(renderer);

    // Освобождаем буфер пикселей
    delete[] pixels;

    // Сообщаем LVGL, что сброс завершен
    lv_disp_flush_ready(disp_drv);
}

// Инициализация LVGL с SDL
void lvgl_init() {
    // Инициализация SDL
```

Продолжение листинга А.2

```
  if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        printf("Ошибка SDL_Init: %s\n", SDL_GetError());
        return;
    }

    // Создание окна
    window = SDL_CreateWindow("ЭГУР User Interface",
                              SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
                              SCREEN_WIDTH, SCREEN_HEIGHT, 0);
    if (window == NULL) {
        printf("Ошибка SDL_CreateWindow: %s\n", SDL_GetError());
        SDL_Quit();
        return;
    }

    // Создание рендерера
    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
    if (renderer == NULL) {
        printf("Ошибка SDL_CreateRenderer: %s\n", SDL_GetError());
        SDL_DestroyWindow(window);
        SDL_Quit();
        return;
    }

    // Создание текстуры
    texture = SDL_CreateTexture(renderer, SDL_PIXELFORMAT_ARGB8888,
                                          SDL_TEXTUREACCESS_STATIC,  SCREEN_WIDTH,
SCREEN_HEIGHT);
    if (texture == NULL) {
        printf("Ошибка SDL_CreateTexture: %s\n", SDL_GetError());
        SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);
        SDL_Quit();
        return;
    }
```

Продолжение листинга А.2

```
    // Инициализация LVGL
    lv_init();
    // Инициализация буфера дисплея
    lv_disp_draw_buf_init(&disp_buf, buf, NULL, SCREEN_WIDTH * 10);
    // Инициализация драйвера дисплея
    lv_disp_drv_init(&disp_drv);
    disp_drv.draw_buf = &disp_buf;
    disp_drv.flush_cb = sdl_flush_cb;
    disp_drv.hor_res = SCREEN_WIDTH;
    disp_drv.ver_res = SCREEN_HEIGHT;
    // Регистрация драйвера дисплея
    lv_disp_drv_register(&disp_drv);
}

#else
// Для микроконтроллера с TFT_eSPI
#include <TFT_eSPI.h>

// Объект TFT
TFT_eSPI tft = TFT_eSPI();

// Буфер дисплея
static lv_disp_draw_buf_t disp_buf;
static lv_color_t buf[SCREEN_WIDTH * 5]; // Уменьшено с 10 до 5 строк

// Драйвер дисплея LVGL
static lv_disp_drv_t disp_drv;

// TFT callback сброса для LVGL
static void tft_flush_cb(lv_disp_drv_t* disp_drv, const lv_area_t* area,
lv_color_t* color_p) {
    uint32_t w = (area->x2 - area->x1 + 1);
    uint32_t h = (area->y2 - area->y1 + 1);
    tft.setAddrWindow(area->x1, area->y1, w, h);
    tft.pushColors((uint16_t*)&color_p->full, w * h, true);
```

Окончание листинга А.2

```
    lv_disp_flush_ready(disp_drv);
}
// Инициализация LVGL с TFT_eSPI
void lvgl_init() {
    // Инициализация TFT
    tft.begin();
    tft.setRotation(0); // Портретный режим
    tft.fillScreen(TFT_BLACK);
    // Инициализация LVGL
    lv_init();
    // Инициализация буфера дисплея
    lv_disp_draw_buf_init(&disp_buf, buf, NULL, SCREEN_WIDTH * 5);

    // Инициализация драйвера дисплея
    lv_disp_drv_init(&disp_drv);
    disp_drv.draw_buf = &disp_buf;
    disp_drv.flush_cb = tft_flush_cb;
    disp_drv.hor_res = SCREEN_WIDTH;
    disp_drv.ver_res = SCREEN_HEIGHT;

    // Регистрация драйвера дисплея
    lv_disp_drv_register(&disp_drv);
}
#endif

// Обработчик тиков LVGL
void lvgl_tick_handler(void* param) {
    (void)param;
    lv_tick_inc(5);
}

// Увеличение тика LVGL
void lvgl_tick_inc(int period_ms) {
    lv_tick_inc(period_ms);
}
```

Листинг А.3 - Управление данными параметров

```
#include "data_manager.h"
#include "config.h"
#include "common.h"  // For millis() in PC mode
#include <cstring>
#include <cstdio>
#include <cmath>    // For sin()
#include <cstdlib>  // For rand()

#ifndef __PC__
#include <Arduino.h> // For millis() in MCU mode
#endif

// Имена параметров
const char* param_names[NUM_PARAMS] = {
    "Давление",
    "Температура",
    "Ток",
    "Угол",
    "Скорость",
    "Напряжение"
};

// Единицы измерения параметров
const char* param_units[NUM_PARAMS] = {
    "бар",
    "°C",
    "А",
    "°",
    "км/ч",
    "В"
};

// Минимальные значения параметров
const float param_mins[NUM_PARAMS] = {
    0.0f,   // Давление
```

Продолжение листинга А.3

```
  -20.0f, // Температура
    0.0f,   // Ток
    -90.0f, // Угол
    0.0f,   // Скорость
    0.0f    // Напряжение
};

// Максимальные значения параметров
const float param_maxs[NUM_PARAMS] = {
    100.0f, // Давление
    120.0f, // Температура
    20.0f,  // Ток
    90.0f,  // Угол
    200.0f, // Скорость
    24.0f   // Напряжение
};

// Значения параметров
float param_values[NUM_PARAMS] = {
    10.4474f,  // Давление
    25.1543f,  // Температура
    5.0655f,   // Ток
    45.0544f,  // Угол
    60.0786f,  // Скорость
    12.0214f   // Напряжение
};

// Буферы истории параметров
static float param_history[NUM_PARAMS][DATA_HISTORY_SIZE];

// Обновление данных ЭГУ
void update_egu_data() {
    static uint32_t last_update = 0;
    uint32_t now = millis();
```

Продолжение листинга А.3

```
    // Обновляем данные каждые DATA_UPDATE_INTERVAL миллисекунд
    if (now - last_update >= DATA_UPDATE_INTERVAL) {
        last_update = now;

        #ifdef __PC__
        // Для симуляции на ПК генерируем случайные колебания параметров
        for (int i = 0; i < NUM_PARAMS; i++) {
            // Малые случайные колебания
            float fluctuation = ((float)rand() / RAND_MAX * 2.0f - 1.0f) * 0.5f;
            param_values[i] += fluctuation;

            // Ограничиваем значения в пределах допустимого
            switch (i) {
                case PARAM_PRESSURE:
                    if (param_values[i] < 0) param_values[i] = 0;
                     if (param_values[i] > PARAM_PRESSURE_MAX) param_values[i] =
PARAM_PRESSURE_MAX;
                    break;
                case PARAM_TEMPERATURE:
                     if (param_values[i] < PARAM_TEMPERATURE_MIN) param_values[i]
= PARAM_TEMPERATURE_MIN;
                     if (param_values[i] > PARAM_TEMPERATURE_MAX) param_values[i]
= PARAM_TEMPERATURE_MAX;
                    break;
                case PARAM_CURRENT:
                    if (param_values[i] < 0) param_values[i] = 0;
                      if (param_values[i] > PARAM_CURRENT_MAX) param_values[i] =
PARAM_CURRENT_MAX;
                    break;
                case PARAM_ANGLE:
                     if (param_values[i] < PARAM_ANGLE_MIN) param_values[i] =
PARAM_ANGLE_MIN;
                     if (param_values[i] > PARAM_ANGLE_MAX) param_values[i] =
PARAM_ANGLE_MAX;
                break;
```

Продолжение листинга А.3

```
        case PARAM_SPEED:
                if (param_values[i] < 0) param_values[i] = 0;
                     if (param_values[i] > PARAM_SPEED_MAX) param_values[i] =
PARAM_SPEED_MAX;
                break;
            case PARAM_VOLTAGE:
                if (param_values[i] < 0) param_values[i] = 0;
                    if (param_values[i] > PARAM_VOLTAGE_MAX) param_values[i] =
PARAM_VOLTAGE_MAX;
                break;
        }

        // Обновляем историю значений параметра
        for (int j = 0; j < DATA_HISTORY_SIZE - 1; j++) {
            param_history[i][j] = param_history[i][j + 1];
        }
        param_history[i][DATA_HISTORY_SIZE - 1] = param_values[i];
    }
    #else
    // Для реального оборудования считываем значения с датчиков
         param_values[PARAM_PRESSURE]    =    analogRead(PRESSURE_SENSOR_PIN)    *
PARAM_PRESSURE_MAX / 1023.0f;
         param_values[PARAM_TEMPERATURE]    =    analogRead(TEMP_SENSOR_PIN)    *
(PARAM_TEMPERATURE_MAX - PARAM_TEMPERATURE_MIN) / 1023.0f + PARAM_TEMPERATURE_MIN;
          param_values[PARAM_CURRENT]    =    analogRead(CURRENT_SENSOR_PIN)    *
PARAM_CURRENT_MAX / 1023.0f;
    param_values[PARAM_ANGLE] = analogRead(ANGLE_SENSOR_PIN) * (PARAM_ANGLE_MAX -
PARAM_ANGLE_MIN) / 1023.0f + PARAM_ANGLE_MIN;
    param_values[PARAM_SPEED] = analogRead(SPEED_SENSOR_PIN) * PARAM_SPEED_MAX /
1023.0f;
          param_values[PARAM_VOLTAGE]    =    analogRead(VOLTAGE_SENSOR_PIN)    *
PARAM_VOLTAGE_MAX / 1023.0f;

    // Обновляем историю значений параметров
```

Окончание листинга А.3

```
  for (int i = 0; i < NUM_PARAMS; i++) {
        for (int j = 0; j < DATA_HISTORY_SIZE - 1; j++) {
            param_history[i][j] = param_history[i][j + 1];
        }
        param_history[i][DATA_HISTORY_SIZE - 1] = param_values[i];
    }
    #endif
}
```

Листинг А.4 - Обработка энкодеров

```
#include "encoders.h"
#include "config.h"

#ifdef __PC__
#include <SDL2/SDL.h>
#else
#include <Arduino.h>
#endif

// Время антидребезга в миллисекундах
#define DEBOUNCE_TIME 50

// Инициализация энкодера
void encoder_init(Encoder* encoder, int pin_a, int pin_b, int sw_pin) {
    encoder->pin_a = pin_a;
    encoder->pin_b = pin_b;
    encoder->sw_pin = sw_pin;
    encoder->last_state = 0;
    encoder->direction = 0;
    encoder->button_pressed = false;
    encoder->button_state = false;
    encoder->last_debounce_time = 0;

    #ifndef __PC__
    // Установка режимов пинов для микроконтроллера
```

Продолжение листинга А.4

```
    pinMode(pin_a, INPUT_PULLUP);
    pinMode(pin_b, INPUT_PULLUP);
    pinMode(sw_pin, INPUT_PULLUP);
    #endif
}

// Считывание состояния энкодера
void encoder_read(Encoder* encoder) {
    #ifdef __PC__
    // В режиме ПК считывание энкодера обрабатывается в главном цикле через события
SDL
    #else
    // Считываем состояния пинов энкодера
    int a_state = digitalRead(encoder->pin_a);
    int b_state = digitalRead(encoder->pin_b);

    // Определяем направление вращения
    if (a_state != encoder->last_state) {
        encoder->direction = (a_state == b_state) ? 1 : -1;
        encoder->last_state = a_state;
    } else {
        encoder->direction = 0;
    }

    // Считываем состояние кнопки с антидребезгом
    uint32_t current_time = millis();
    bool reading = digitalRead(encoder->sw_pin) == LOW;

    if (reading != encoder->button_state) {
        encoder->last_debounce_time = current_time;
    }

    if ((current_time - encoder->last_debounce_time) > DEBOUNCE_TIME) {
        if (reading && !encoder->button_state) {
            encoder->button_pressed = true;
```

Окончание листинга А.4

```
        } else {
            encoder->button_pressed = false;
        }
        encoder->button_state = reading;
    }
    #endif
}
```

Листинг А.5 - Обработка ошибок

```
#include "error_handler.h"
#include "data_manager.h"
#include "config.h"
#include "lvgl.h"
#include <cstdio>

// Текущий код ошибки
static uint8_t current_error = ERR_NONE;

// Сообщения об ошибках
static const char* error_messages[] = {
    "Нет ошибки",
    "Ошибка датчика давления",
    "Ошибка датчика температуры",
    "Ошибка датчика тока",
    "Ошибка датчика угла",
    "Ошибка датчика скорости",
    "Ошибка датчика напряжения",
    "Ошибка связи",
    "Ошибка памяти"
};

// Инициализация обработчика ошибок
void error_handler_init() {
    current_error = ERR_NONE;
}
```

Продолжение листинга А.5

```
// Проверка ошибок датчиков
void check_errors() {
    // Получаем текущие значения параметров
    float pressure = get_param_value(PARAM_PRESSURE);
    float temp = get_param_value(PARAM_TEMPERATURE);
    float current = get_param_value(PARAM_CURRENT);
    float angle = get_param_value(PARAM_ANGLE);
    float speed = get_param_value(PARAM_SPEED);
    float voltage = get_param_value(PARAM_VOLTAGE);

    // Проверяем ошибки датчиков (значения вне допустимого диапазона)
    if (pressure < 0 || pressure > PARAM_PRESSURE_MAX * 1.1f) {
        set_error(ERR_SENSOR_PRESSURE);
    } else {
        clear_error(ERR_SENSOR_PRESSURE);
    }

     if (temp < PARAM_TEMPERATURE_MIN * 1.1f || temp > PARAM_TEMPERATURE_MAX *
1.1f) {
        set_error(ERR_SENSOR_TEMPERATURE);
    } else {
        clear_error(ERR_SENSOR_TEMPERATURE);
    }

    if (current < 0 || current > PARAM_CURRENT_MAX * 1.1f) {
        set_error(ERR_SENSOR_CURRENT);
    } else {
        clear_error(ERR_SENSOR_CURRENT);
    }

    if (angle < PARAM_ANGLE_MIN * 1.1f || angle > PARAM_ANGLE_MAX * 1.1f) {
        set_error(ERR_SENSOR_ANGLE);
    } else {
        clear_error(ERR_SENSOR_ANGLE);
    }
```

Продолжение листинга А.5

```
    if (speed < 0 || speed > PARAM_SPEED_MAX * 1.1f) {
        set_error(ERR_SENSOR_SPEED);
    } else {
        clear_error(ERR_SENSOR_SPEED);
    }

    if (voltage < 0 || voltage > PARAM_VOLTAGE_MAX * 1.1f) {
        set_error(ERR_SENSOR_VOLTAGE);
    } else {
        clear_error(ERR_SENSOR_VOLTAGE);
    }
}

// Установка ошибки
void set_error(uint8_t error_code) {
    current_error |= (1 << error_code);
}

// Очистка ошибки
void clear_error(uint8_t error_code) {
    current_error &= ~(1 << error_code);
}

// Получение текущего кода ошибки
uint8_t get_error() {
    return current_error;
}

// Проверка, активна ли ошибка
bool is_error_active(uint8_t error_code) {
    return (current_error & (1 << error_code)) != 0;
}

// Получение сообщения об ошибке по коду
const char* get_error_message(uint8_t error_code) {
```

Продолжение листинга А.5

```
    if (error_code < sizeof(error_messages) / sizeof(error_messages[0])) {
        return error_messages[error_code];
    }
    return "Неизвестная ошибка";
}

// Отображение экрана ошибок
void show_error_screen() {
    // Очистка экрана
    lv_obj_clean(lv_scr_act());

    // Создание заголовка ошибки
    lv_obj_t* title = lv_label_create(lv_scr_act());
    lv_label_set_text(title, "СИСТЕМНАЯ ОШИБКА");
    lv_obj_align(title, LV_ALIGN_TOP_MID, 0, 20);
    lv_obj_set_style_text_color(title, lv_color_hex(0xFF0000), LV_PART_MAIN);

    // Создание списка ошибок
    lv_obj_t* error_list = lv_list_create(lv_scr_act());
    lv_obj_set_size(error_list, 280, 160);
    lv_obj_align(error_list, LV_ALIGN_CENTER, 0, 10);

    // Добавляем активные ошибки в список
    for (int i = 1; i < 9; i++) {
        if (is_error_active(i)) {
            lv_list_add_text(error_list, get_error_message(i));
        }
    }

    // Инструкция для пользователя
    lv_obj_t* instructions = lv_label_create(lv_scr_act());
    lv_label_set_text(instructions, "Нажмите любую кнопку для возврата");
    lv_obj_align(instructions, LV_ALIGN_BOTTOM_MID, 0, -10);
}
```

Окончание листинга А.5

```
// Показ сообщения об ошибке
void show_error_message(uint8_t error_code) {
    // Кнопки для окна сообщения
    static const char* btns[] = {"OK", ""};

    // Создание окна сообщения
             lv_obj_t*      mbox      =      lv_msgbox_create(NULL,      "Ошибка",
get_error_message(error_code), btns, false);
                           lv_obj_set_style_text_color(lv_msgbox_get_title(mbox),
lv_color_hex(0xFF0000), LV_PART_MAIN);
    lv_obj_center(mbox);

    // Автоматическое закрытие через 3 секунды
    lv_timer_t* timer = lv_timer_create([](lv_timer_t* timer) {
        lv_obj_t* mbox = (lv_obj_t*)timer->user_data;
        lv_msgbox_close(mbox);
        lv_timer_del(timer);
    }, 3000, mbox);
}
```

Листинг А.6 - Коммуникационный интерфейс

```
#include "communication.h"
#include "data_manager.h"
#include "config.h"  // Для NUM_PARAMS и DATA_HISTORY_SIZE
#include <cstdint>
#include <cstring>  // Для memcpy

#ifdef __PC__
// Режим симуляции на ПК через последовательный порт
#include <iostream>
#include <windows.h>  // Для работы с COM-портом в Windows

HANDLE hSerial = INVALID_HANDLE_VALUE;

void comm_init() {
```

Продолжение листинга А.6

```
   // Открываем COM порт (используйте тот же порт, что и COMPIM)
    hSerial = CreateFile("COM3",  // Измените на нужный порт
                         GENERIC_READ | GENERIC_WRITE,
                         0,
                         NULL,
                         OPEN_EXISTING,
                         FILE_ATTRIBUTE_NORMAL,
                         NULL);

    if (hSerial == INVALID_HANDLE_VALUE) {
        std::cout << "Ошибка открытия последовательного порта" << std::endl;
        return;
    }
    // Настройка последовательного порта
    DCB dcbSerialParams = {0};
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

    if (!GetCommState(hSerial, &dcbSerialParams)) {
          std::cout << "Ошибка получения состояния последовательного порта" <<
std::endl;
        CloseHandle(hSerial);
        return;
    }

    dcbSerialParams.BaudRate = CBR_115200;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;

    if (!SetCommState(hSerial, &dcbSerialParams)) {
          std::cout << "Ошибка установки состояния последовательного порта" <<
std::endl;
        CloseHandle(hSerial);
        return;
    }
```

Продолжение листинга А.6

```
    std::cout << "Последовательный порт инициализирован" << std::endl;
}
#else
#include <Arduino.h>
#endif

// Буфер для входящих данных
#define COMM_BUFFER_SIZE 64
uint8_t comm_buffer[COMM_BUFFER_SIZE];
uint8_t comm_buffer_index = 0;

// Внешние переменные и функции
extern bool diagnostic_mode;
extern void run_all_diagnostics();
extern uint8_t system_error_code;
extern uint8_t get_selected_parameter();

// Обработка входящих команд
#ifdef __PC__
void comm_process() {
    if (hSerial == INVALID_HANDLE_VALUE) return;

    DWORD bytesRead = 0;
    DWORD errors;
    COMSTAT status;

    ClearCommError(hSerial, &errors, &status);

    if (status.cbInQue > 0) {
        uint8_t byte;

        while (status.cbInQue > 0 && comm_buffer_index < COMM_BUFFER_SIZE) {
            ReadFile(hSerial, &byte, 1, &bytesRead, NULL);

            // Простой протокол: байт команды + данные
```

Продолжение листинга А.6

```
            if (comm_buffer_index == 0 && byte != 0xAA) {
                // Неверный стартовый байт, игнорируем
                continue;
            }

            // Сохраняем байт в буфер
            comm_buffer[comm_buffer_index++] = byte;

            // Проверяем конец команды (байт 0xFF)
            if (byte == 0xFF || comm_buffer_index >= COMM_BUFFER_SIZE) {
                comm_parse_command(comm_buffer, comm_buffer_index);
                comm_buffer_index = 0;
            }

            ClearCommError(hSerial, &errors, &status);
        }
    }
}

// Отправка данных в Proteus через последовательный порт
void serial_write(uint8_t* data, uint16_t length) {
    if (hSerial == INVALID_HANDLE_VALUE) return;

    DWORD bytesWritten = 0;
    WriteFile(hSerial, data, length, &bytesWritten, NULL);
}
#else
void comm_process() {
    while (Serial.available() > 0) {
        uint8_t byte = Serial.read();

        if (comm_buffer_index == 0 && byte != 0xAA) {
            continue;
        }
```

Продолжение листинга А.6

```
        comm_buffer[comm_buffer_index++] = byte;

        if (byte == 0xFF || comm_buffer_index >= COMM_BUFFER_SIZE) {
            comm_parse_command(comm_buffer, comm_buffer_index);
            comm_buffer_index = 0;
        }
    }
}
#endif

// Разбор и выполнение команды
void comm_parse_command(uint8_t* buffer, uint8_t length) {
    if (length < 3) return;

    if (buffer[0] != 0xAA) return;

    uint8_t cmd = buffer[1];

    switch (cmd) {
        case CMD_GET_PARAM:
            if (length >= 4) {
                uint8_t param_id = buffer[2];
                comm_send_param(param_id);
            }
            break;
        case CMD_SET_PARAM:
            if (length >= 8) {
                uint8_t param_id = buffer[2];
                float value;
                memcpy(&value, &buffer[3], sizeof(float));
                set_param_value(param_id, value);
            }
            break;
        case CMD_GET_ALL_PARAMS:
            comm_send_all_params();
```

Продолжение листинга А.6

```
            break;
        case CMD_GET_HISTORY:
            if (length >= 4) {
                uint8_t param_id = buffer[2];
                comm_send_history(param_id);
            }
            break;
        case CMD_RUN_DIAG:
            diagnostic_mode = true;
            run_all_diagnostics();
            break;
        case CMD_EXIT_DIAG:
            diagnostic_mode = false;
            break;
    }
}
// Отправка значения параметра
void comm_send_param(uint8_t param_id) {
    if (param_id >= NUM_PARAMS) return;

    uint8_t response[8];
    float value = get_param_value(param_id);

    response[0] = 0xAA;
    response[1] = CMD_GET_PARAM;
    response[2] = param_id;
    memcpy(&response[3], &value, sizeof(float));
    response[7] = 0xFF;

    #ifdef __PC__
    serial_write(response, 8);
    #else
    Serial.write(response, 8);
    #endif
}
```

Продолжение листинга А.6

```
// Отправка всех параметров
void comm_send_all_params() {
    #ifndef __PC__
    uint8_t response[3 + NUM_PARAMS * 5];

    response[0] = 0xAA;
    response[1] = CMD_GET_ALL_PARAMS;
    response[2] = NUM_PARAMS;

    for (int i = 0; i < NUM_PARAMS; i++) {
        response[3 + i * 5] = i;
        float value = get_param_value(i);
        memcpy(&response[4 + i * 5], &value, sizeof(float));
    }

    response[3 + NUM_PARAMS * 5 - 1] = 0xFF;

    Serial.write(response, 3 + NUM_PARAMS * 5);
    #endif
}

// Отправка истории параметра
void comm_send_history(uint8_t param_id) {
    #ifndef __PC__
    if (param_id >= NUM_PARAMS) return;

    float* history = get_param_history(param_id);

    uint16_t response_size = 4 + DATA_HISTORY_SIZE * sizeof(float);
    uint8_t* response = new uint8_t[response_size];

    response[0] = 0xAA;
    response[1] = CMD_GET_HISTORY;
    response[2] = param_id;
    response[3] = DATA_HISTORY_SIZE;
```

Окончание листинга А.6

```
    memcpy(&response[4], history, DATA_HISTORY_SIZE * sizeof(float));

    response[response_size - 1] = 0xFF;

    Serial.write(response, response_size);
    delete[] response;
    #endif
}
```

Листинг А.7 - Общие функции UI

```
#include "ui/ui.h"
#include "data_manager.h"
#include "config.h"
#include "common.h"
#include "lvgl.h"
#include <cstdio>

// Внешние объявления функций для предотвращения множественных определений
extern void create_main_screen();
extern void update_main_screen();
extern void create_edit_screen_content(int param_id);
extern void update_edit_screen();
extern void create_chart_screen(int param_id);
extern void update_chart_screen();

// Глобальные переменные UI
int edit_param_id = 0;
float edit_value = 0.0f;

// Обработка подтверждения (сохранить или отменить)
void handle_confirmation(bool save) {
    if (save) {
        // Сохраняем отредактированное значение
        save_parameter_value(edit_param_id, edit_value);
    }
```

Окончание листинга А.7

```
    // Возврат на главный экран
    extern ScreenType current_screen;
    current_screen = SCREEN_MAIN;
    lv_obj_clean(lv_scr_act());
    create_main_screen();
}
```

Листинг А.8 - Главный экран

```
#include "ui/ui.h"
#include "data_manager.h"
#include "config.h"
#include "common.h"
#include "lvgl.h"
#include <cstdio>

// Элементы UI главного экрана
static lv_obj_t* param_labels[NUM_PARAMS];
static lv_obj_t* param_value_labels[NUM_PARAMS];
// Выбранный параметр на главном экране
int selected_param = 0;

// Создание главного экрана
void create_main_screen() {
    printf("Создание компонентов главного экрана...\n");

    // Очистка экрана
    lv_obj_clean(lv_scr_act());

    // Создаем главный контейнер с вертикальным расположением (3 строки)
    lv_obj_t* cont = lv_obj_create(lv_scr_act());
    lv_obj_set_size(cont, SCREEN_WIDTH, SCREEN_HEIGHT);
    lv_obj_align(cont, LV_ALIGN_TOP_MID, 0, 0);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN); // Вертикальный флекс

    lv_obj_set_style_pad_all(cont, 5, 0);
```

Продолжение листинга А.8

```
    // 6 параметров, 3 строки, 2 колонки
    int params_per_row = 2;
    int rows = 3;

    // Массивы для хранения элементов UI
    static lv_obj_t* param_labels[NUM_PARAMS];
    static lv_obj_t* param_value_labels[NUM_PARAMS];

    for (int row = 0; row < rows; row++) {
        // Создаем строку - контейнер с горизонтальным флексом
        lv_obj_t* row_cont = lv_obj_create(cont);
        lv_obj_set_size(row_cont, SCREEN_WIDTH - 10, 70);
        lv_obj_set_flex_flow(row_cont, LV_FLEX_FLOW_ROW);
        lv_obj_set_style_pad_all(row_cont, 4, 0);

        for (int col = 0; col < params_per_row; col++) {
            int idx = row * params_per_row + col;
            if (idx >= NUM_PARAMS) break;

            // Контейнер для одного параметра (название + значение)
            lv_obj_t* param_cont = lv_obj_create(row_cont);
            lv_obj_set_size(param_cont, (SCREEN_WIDTH - 10)/2 , 70);
            lv_obj_set_flex_flow(param_cont, LV_FLEX_FLOW_COLUMN);
            lv_obj_set_style_pad_all(param_cont, 0, 0);

            // Метка названия параметра
            param_labels[idx] = lv_label_create(param_cont);
            lv_label_set_text(param_labels[idx], get_param_name(idx));
            lv_obj_set_width(param_labels[idx], ((SCREEN_WIDTH - 10)) / 2);

            // Метка значения параметра
            param_value_labels[idx] = lv_label_create(param_cont);
            char buf[32];
                snprintf(buf, sizeof(buf), "%.*g %s", 6, get_param_value(idx),
get_param_unit(idx));
```

Продолжение листинга А.8

```
            lv_label_set_text(param_value_labels[idx], buf);
            lv_obj_set_width(param_value_labels[idx], ((SCREEN_WIDTH - 10)) / 2);
            lv_obj_align(param_value_labels[idx], LV_ALIGN_RIGHT_MID, 0, 0);

            // Подсветка выбранного параметра
            if (idx == selected_param) {
                                        lv_obj_set_style_bg_color(param_cont,
lv_palette_main(LV_PALETTE_BLUE), LV_PART_MAIN);
                lv_obj_set_style_text_color(param_labels[idx], lv_color_white(),
LV_PART_MAIN);
                            lv_obj_set_style_text_color(param_value_labels[idx],
lv_color_white(), LV_PART_MAIN);
            } else {
                  lv_obj_set_style_bg_color(param_cont, lv_color_make(201, 201,
201), LV_PART_MAIN);
                lv_obj_set_style_text_color(param_labels[idx], lv_color_black(),
LV_PART_MAIN);
                            lv_obj_set_style_text_color(param_value_labels[idx],
lv_color_black(), LV_PART_MAIN);
            }
        }
    }

    printf("Главный экран успешно создан\n");
}

// Обновление главного экрана
void update_main_screen() {
    // Обновляем значения параметров
    for (int i = 0; i < NUM_PARAMS; i++) {
        if (param_value_labels[i] != NULL) {
            char buf[32];
                  snprintf(buf,  sizeof(buf),  "%.2f  %s",  get_param_value(i),
get_param_unit(i));
```

Продолжение листинга А.8

```
            lv_label_set_text(param_value_labels[i], buf);
        }
    }
}

// Выбор параметра
void select_parameter(int direction) {
    lv_obj_t* cont = lv_obj_get_child(lv_scr_act(), 0); // главный контейнер
    if (!cont) return;

    int old_idx = selected_param;

    int rows = 3;
    int cols = 2;

    int old_row = old_idx / cols;
    int old_col = old_idx % cols;

    lv_obj_t* row_cont = lv_obj_get_child(cont, old_row);
    if (row_cont) {
        lv_obj_t* param_cont = lv_obj_get_child(row_cont, old_col);
        if (param_cont) {
             lv_obj_set_style_bg_color(param_cont, lv_color_make(201, 201, 201),
LV_PART_MAIN);
            lv_obj_t* name_label = lv_obj_get_child(param_cont, 0);
            lv_obj_t* value_label = lv_obj_get_child(param_cont, 1);
                      if   (name_label)   lv_obj_set_style_text_color(name_label,
lv_color_black(), LV_PART_MAIN);
                    if   (value_label)   lv_obj_set_style_text_color(value_label,
lv_color_black(), LV_PART_MAIN);
        }
    }

    selected_param = (selected_param + direction + NUM_PARAMS) % NUM_PARAMS;
```

Окончание листинга А.8

```
    int new_row = selected_param / cols;
    int new_col = selected_param % cols;

    row_cont = lv_obj_get_child(cont, new_row);
    if (row_cont) {
        lv_obj_t* param_cont = lv_obj_get_child(row_cont, new_col);
        if (param_cont) {
                                            lv_obj_set_style_bg_color(param_cont,
lv_palette_main(LV_PALETTE_BLUE), LV_PART_MAIN);
            lv_obj_t* name_label = lv_obj_get_child(param_cont, 0);
            lv_obj_t* value_label = lv_obj_get_child(param_cont, 1);
                      if   (name_label)   lv_obj_set_style_text_color(name_label,
lv_color_white(), LV_PART_MAIN);
                    if   (value_label)   lv_obj_set_style_text_color(value_label,
lv_color_white(), LV_PART_MAIN);
        }
    }

            printf("Выбран     параметр:     %d     (%s)\n",     selected_param,
get_param_name(selected_param));
}

// Получить выбранный параметр
uint8_t get_selected_parameter() {
    return selected_param;
}
```

Листинг А.9 - Экран редактирования

```
#include "ui/ui.h"
#include "data_manager.h"
#include "config.h"
#include "common.h"
#include "lvgl.h"
#include <cstdio>
```

Продолжение листинга А.9

```
// Внешние переменные из ui.cpp
extern int edit_param_id;
extern float edit_value;

// Элементы UI экрана редактирования
static lv_obj_t* param_name_label;
static lv_obj_t* param_value_label;
static lv_obj_t* slider;
static lv_obj_t* save_btn;
static lv_obj_t* cancel_btn;

// Коллбек кнопки "Сохранить"
static void save_button_cb(lv_event_t* e) {
    LV_UNUSED(e);
    save_parameter_value(edit_param_id, edit_value);
    handle_confirmation(true); // Сохраняем и возвращаемся
}

// Коллбек кнопки "Отмена"
static void cancel_button_cb(lv_event_t* e) {
    LV_UNUSED(e);
    handle_confirmation(false); // Отменяем и возвращаемся
}

// Создание содержимого экрана редактирования
void create_edit_screen_content(int param_id) {
     printf("Создание экрана редактирования для параметра %d (%s)\n", param_id,
get_param_name(param_id));

    // Установка глобальных переменных
    edit_param_id = param_id;
    edit_value = get_param_value(param_id);

    // Заголовок
    lv_obj_t* title = lv_label_create(lv_scr_act());
```

Продолжение листинга А.9

```
lv_label_set_text_fmt(title, "Редактирование %s", get_param_name(param_id));
lv_obj_align(title, LV_ALIGN_TOP_MID, 0, 10);

// Метка с названием параметра
param_name_label = lv_label_create(lv_scr_act());
lv_label_set_text(param_name_label, get_param_name(param_id));
lv_obj_align(param_name_label, LV_ALIGN_TOP_MID, 0, 40);

// Метка с текущим значением параметра
param_value_label = lv_label_create(lv_scr_act());
char buf[16];
snprintf(buf, sizeof(buf), "%.2f %s", edit_value, get_param_unit(param_id));
lv_label_set_text(param_value_label, buf);
lv_obj_align(param_value_label, LV_ALIGN_TOP_MID, 0, 70);

// Создание слайдера для изменения значения
slider = lv_slider_create(lv_scr_act());
lv_obj_set_width(slider, SCREEN_WIDTH - 40);
lv_obj_align(slider, LV_ALIGN_CENTER, 0, 0);
lv_slider_set_range(slider,
                    (int)(get_param_min(param_id) * 100),
                    (int)(get_param_max(param_id) * 100));
lv_slider_set_value(slider, (int)(edit_value * 100), LV_ANIM_OFF);

// Кнопка "Сохранить"
save_btn = lv_btn_create(lv_scr_act());
lv_obj_set_size(save_btn, 100, 40);
lv_obj_align(save_btn, LV_ALIGN_BOTTOM_LEFT, 20, -20);
lv_obj_t* save_label = lv_label_create(save_btn);
lv_label_set_text(save_label, "Сохранить");
lv_obj_center(save_label);
lv_obj_add_event_cb(save_btn, save_button_cb, LV_EVENT_CLICKED, NULL);

// Кнопка "Отмена"
cancel_btn = lv_btn_create(lv_scr_act());
```

Продолжение листинга А.9

```
  lv_obj_set_size(cancel_btn, 100, 40);
    lv_obj_align(cancel_btn, LV_ALIGN_BOTTOM_RIGHT, -20, -20);
    lv_obj_t* cancel_label = lv_label_create(cancel_btn);
    lv_label_set_text(cancel_label, "Отмена");
    lv_obj_center(cancel_label);
    lv_obj_add_event_cb(cancel_btn, cancel_button_cb, LV_EVENT_CLICKED, NULL);

    printf("Экран редактирования создан\n");
}

// Обновление экрана редактирования
void update_edit_screen() {
     static float last_value = -999.0f;  // Инициализация заведомо невозможным
значением

    // Обновляем только если значение изменилось
    if (edit_value != last_value) {
        // Обновляем метку с значением
        char buf[16];
                 snprintf(buf,   sizeof(buf),   "%.*g   %s",   6,   edit_value,
get_param_unit(edit_param_id));
        lv_label_set_text(param_value_label, buf);

        // Обновляем слайдер
        lv_slider_set_value(slider, (int)(edit_value * 100), LV_ANIM_OFF);

           printf("Обновлено  редактируемое  значение:  %.2f  %s\n",  edit_value,
get_param_unit(edit_param_id));

        // Сохраняем последнее значение
        last_value = edit_value;
    }
}

// Корректировка значения параметра
```

Окончание листинга А.9

```
void adjust_parameter_value(int direction) {
    float min_val = get_param_min(edit_param_id);
    float max_val = get_param_max(edit_param_id);
    float range = max_val - min_val;
    float step = range / 100.0f;  // 100 шагов по диапазону

    edit_value += direction * step;

    if (edit_value < min_val) edit_value = min_val;
    if (edit_value > max_val) edit_value = max_val;

    update_edit_screen();

    printf("Параметр %d (%s) изменён на %.2f %s\n",
           edit_param_id, get_param_name(edit_param_id),
           edit_value, get_param_unit(edit_param_id));
}
```

Листинг А.10 - Экран графиков

```
#include "ui/ui.h"
#include "data_manager.h"
#include "config.h"
#include "common.h"
#include "lvgl.h"
#include <cstdio>

// Элементы UI экрана графика
static lv_obj_t* chart;
static lv_chart_series_t* chart_series;
static int chart_param_id;

// Создание экрана графика
void create_chart_screen(int param_id) {
      printf("Создание экрана графика для параметра %d (%s)n", param_id,
get_param_name(param_id));
```

Продолжение листинга А.10

```
    chart_param_id = param_id;

    // Заголовок
    lv_obj_t* title = lv_label_create(lv_scr_act());
    lv_label_set_text_fmt(title, "График %s", get_param_name(param_id));
    lv_obj_align(title, LV_ALIGN_TOP_MID, 0, 10);

    // Создание графика
    chart = lv_chart_create(lv_scr_act());
    lv_obj_set_size(chart, SCREEN_WIDTH - 20, SCREEN_HEIGHT - 80);
    lv_obj_align(chart, LV_ALIGN_CENTER, 0, 0);
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE);
    lv_chart_set_point_count(chart, DATA_HISTORY_SIZE);
    lv_chart_set_range(chart, LV_CHART_AXIS_PRIMARY_Y,
                      (lv_coord_t)get_param_min(param_id),
                      (lv_coord_t)get_param_max(param_id));

    // Добавление серии данных
     chart_series = lv_chart_add_series(chart, lv_palette_main(LV_PALETTE_RED),
LV_CHART_AXIS_PRIMARY_Y);

    // Заполнение графика данными из истории
    float* history = get_param_history(param_id);
    for (int i = 0; i < DATA_HISTORY_SIZE; i++) {
        lv_chart_set_next_value(chart, chart_series, (lv_coord_t)history[i]);
    }

    // Инструкция по управлению
    lv_obj_t* instructions = lv_label_create(lv_scr_act());
    lv_label_set_text(instructions, "Энкодер 1: масштаб X | Энкодер 2: масштаб Y
| Любая кнопка: назад");
    lv_obj_align(instructions, LV_ALIGN_BOTTOM_MID, 0, -10);

    printf("Экран графика созданп");
}
```

Продолжение листинга А.10

```
// Обновление экрана графика
void update_chart_screen() {
    if (chart != NULL && chart_series != NULL) {
        float* history = get_param_history(chart_param_id);
        if (history != NULL) {
            for (int i = 0; i < DATA_HISTORY_SIZE; i++) {
                                lv_chart_set_value_by_id(chart,   chart_series,   i,
(lv_coord_t)history[i]);
            }
            lv_chart_refresh(chart);
        }
    }
}

// Масштабирование оси X графика
void chart_scale_x(int direction) {
    if (chart != NULL) {
        static int point_count = DATA_HISTORY_SIZE;

        point_count = point_count + direction * 10;
        if (point_count < 10) point_count = 10;
        if (point_count > DATA_HISTORY_SIZE) point_count = DATA_HISTORY_SIZE;

        lv_chart_set_point_count(chart, point_count);
        lv_chart_refresh(chart);
        printf("Масштаб оси X изменён на %d точекn", point_count);
    }
}

// Масштабирование оси Y графика
void chart_scale_y(int direction) {
    if (chart != NULL) {
        static float scale_factor = 1.0f;

        if (direction > 0) {
```

Окончание листинга А.10

```
            scale_factor *= 0.8f;  // Приближение
             if (scale_factor < 0.1f) scale_factor = 0.1f;
        } else {
             scale_factor *= 1.25f;  // Отдаление
             if (scale_factor > 2.0f) scale_factor = 2.0f;
        }

        float param_min = get_param_min(chart_param_id);
        float param_max = get_param_max(chart_param_id);
        float range = param_max - param_min;
        float mid = (param_min + param_max) / 2.0f;

        float new_range = range * scale_factor;
        lv_coord_t new_min = (lv_coord_t)(mid - new_range / 2);
        lv_coord_t new_max = (lv_coord_t)(mid + new_range / 2);

        lv_chart_set_range(chart, LV_CHART_AXIS_PRIMARY_Y, new_min, new_max);
        lv_chart_refresh(chart);

        printf("Масштаб оси Y изменён: от %d до %d (фактор масштаба: %.2f)n",
               (int)new_min, (int)new_max, scale_factor);
    }
}
```

Листинг А.11 - Компонент отображения чисел

```
#include "ui/components/number_display.h"
#include <stdio.h>

// Создание компонента для отображения числа
lv_obj_t* create_number_display(lv_obj_t* parent, int x, int y) {
    lv_obj_t* label = lv_label_create(parent);
    lv_obj_set_pos(label, x, y);
    lv_obj_set_size(label, 100, 30);
    lv_label_set_text(label, "0.00"); // Начальное значение
```

Окончание листинга А.11

```
    return label;
}

// Установка значения в компоненте отображения числа
void set_number_display_value(lv_obj_t* display, float value) {
    char buffer[20];
    sprintf(buffer, "%.2f", value); // Форматирование значения с двумя знаками
после запятой
    lv_label_set_text(display, buffer);
}
```

Листинг А.12 - Реализация для микроконтроллера

```
#include "hal/hal.h"

#ifndef __PC__

#include <Arduino.h>

// Обертка для digitalRead
int digital_read(int pin) {
    return digitalRead(pin);
}
#endif
```

Листинг А.13 - Реализация для SDL (ПК)

```
#include "lvgl.h"
#include "config.h"
#include "encoders.h"

#ifdef __PC__
#include <SDL2/SDL.h>

// Чтение цифрового пина для симуляции в SDL
```

Продолжение листинга А.13

```
int digital_read(int pin) {
    // В режиме SDL симулируем состояния пинов на основе клавиатурного ввода
    static int pin_states[32] = {0}; // Хранение состояний для всех пинов

    return pin_states[pin];
}

// Обработка событий SDL для клавиатурного ввода
void handle_sdl_events() {
    SDL_Event event;
    while (SDL_PollEvent(&event)) {
        if (event.type == SDL_QUIT) {
            exit(0);
        }

        if (event.type == SDL_KEYDOWN) {
            switch (event.key.keysym.sym) {
                case SDLK_a: // Энкодер 1 против часовой
                    encoder1.direction = -1;
                    break;
                case SDLK_d: // Энкодер 1 по часовой
                    encoder1.direction = 1;
                    break;
                case SDLK_s: // Кнопка энкодера 1
                    encoder1.button_pressed = true;
                    break;
                case SDLK_j: // Энкодер 2 против часовой
                    encoder2.direction = -1;
                    break;
                case SDLK_l: // Энкодер 2 по часовой
                    encoder2.direction = 1;
                    break;
                case SDLK_k: // Кнопка энкодера 2
                    encoder2.button_pressed = true;
                    break;
```

Окончание листинга А.13

```
            case SDLK_ESCAPE: // Выход из приложения
                    exit(0);
                    break;
                default:
                    break;
            }
        }
    }
}
#endif
```

# ПРИЛОЖЕНИЕ В. ЗАГОЛОВОЧНЫЕ ФАЙЛЫ

Листинг В.1 - Конфигурация проекта

```
#ifndef CONFIG_H
#define CONFIG_H

// Размер экрана
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240

// Количество параметров
#define NUM_PARAMS 6

// Размер истории данных для каждого параметра
#define DATA_HISTORY_SIZE 100

// Интервал обновления данных (мс)
#define DATA_UPDATE_INTERVAL 1000

// Пины энкодеров (для микроконтроллера)
#define ENCODER_1_PIN_A 2
#define ENCODER_1_PIN_B 3
#define ENCODER_1_SW_PIN 4

#define ENCODER_2_PIN_A 5
#define ENCODER_2_PIN_B 6
#define ENCODER_2_SW_PIN 7

// Пины датчиков (для микроконтроллера)
#define PRESSURE_SENSOR_PIN A0
#define TEMP_SENSOR_PIN A1
#define CURRENT_SENSOR_PIN A2
#define ANGLE_SENSOR_PIN A3
#define SPEED_SENSOR_PIN A4
#define VOLTAGE_SENSOR_PIN A5

// Максимальные и минимальные значения параметров
```

Окончание листинга В.1

```
#define PARAM_PRESSURE_MAX 100.0f
#define PARAM_TEMPERATURE_MIN -20.0f
#define PARAM_TEMPERATURE_MAX 120.0f
#define PARAM_CURRENT_MAX 20.0f
#define PARAM_ANGLE_MIN -90.0f
#define PARAM_ANGLE_MAX 90.0f
#define PARAM_SPEED_MAX 200.0f
#define PARAM_VOLTAGE_MAX 24.0f

// Команды для протокола связи
#define CMD_GET_PARAM 0x01
#define CMD_SET_PARAM 0x02
#define CMD_GET_ALL_PARAMS 0x03
#define CMD_GET_HISTORY 0x04
#define CMD_RUN_DIAG 0x05
#define CMD_EXIT_DIAG 0x06

// Коды ошибок
#define ERR_NONE 0
#define ERR_SENSOR_PRESSURE 1
#define ERR_SENSOR_TEMPERATURE 2
#define ERR_SENSOR_CURRENT 3
#define ERR_SENSOR_ANGLE 4
#define ERR_SENSOR_SPEED 5
#define ERR_SENSOR_VOLTAGE 6
#define ERR_COMMUNICATION 7
#define ERR_MEMORY 8

#endif // CONFIG_H
```

Листинг В.2 - Общие определения

```
#ifndef COMMON_H
#define COMMON_H
#include <stdint.h>
```

Окончание листинга В.2

```
// Типы экранов
typedef enum {
      SCREEN_MAIN,
      SCREEN_EDIT,
      SCREEN_CHART,
      SCREEN_ERROR
} ScreenType;
// Параметры
typedef enum {
      PARAM_PRESSURE = 0,
      PARAM_TEMPERATURE,
      PARAM_CURRENT,
      PARAM_ANGLE,
      PARAM_SPEED,
      PARAM_VOLTAGE
} ParamID;
#endif // COMMON_H
```

Листинг В.3 - Интерфейс работы с энкодерами

```
#ifndef ENCODERS_H
#define ENCODERS_H

#include <stdbool.h>

// Структура энкодера
typedef struct {
      int pin_a;
      int pin_b;
      int sw_pin;
      int last_state;
      int direction;
      bool button_pressed;
      bool button_state;
      unsigned long last_debounce_time;
} Encoder;
```

Окончание листинга В.3

```
// Инициализация энкодера
void encoder_init(Encoder* encoder, int pin_a, int pin_b, int sw_pin);

// Считывание состояния энкодера
void encoder_read(Encoder* encoder);

#endif // ENCODERS_H
```

Листинг В.4 - Интерфейс управления данными

```
#ifndef DATA_MANAGER_H
#define DATA_MANAGER_H

#include <stdint.h>

#define NUM_PARAMS 6
#define DATA_HISTORY_SIZE 100

// Получить имя параметра по ID
const char* get_param_name(int param_id);

// Получить единицу измерения по ID
const char* get_param_unit(int param_id);

// Получить минимальное значение параметра
float get_param_min(int param_id);

// Получить максимальное значение параметра
float get_param_max(int param_id);

// Получить текущее значение параметра
float get_param_value(int param_id);

// Установить значение параметра
void set_param_value(int param_id, float value);
```

Окончание листинга В.4

```
// Получить историю параметра
float* get_param_history(int param_id);

// Обновить данные ЭГУ
void update_egu_data();

// Сохранить значение параметра
void save_parameter_value(int param_id, float value);

#endif // DATA_MANAGER_H
```

Листинг В.5 - Интерфейс обработки ошибок

```
#ifndef ERROR_HANDLER_H
#define ERROR_HANDLER_H

#include <stdint.h>
#include <stdbool.h>

// Инициализация обработчика ошибок
void error_handler_init();

// Проверка ошибок датчиков
void check_errors();

// Установка ошибки
void set_error(uint8_t error_code);

// Очистка ошибки
void clear_error(uint8_t error_code);

// Получение текущего кода ошибки
uint8_t get_error();

// Проверка, активна ли ошибка
bool is_error_active(uint8_t error_code);
```

Окончание листинга В.5

```
// Получение сообщения об ошибке
const char* get_error_message(uint8_t error_code);

// Отображение экрана ошибок
void show_error_screen();

// Показ сообщения об ошибке
void show_error_message(uint8_t error_code);

#endif // ERROR_HANDLER_H
```

Листинг В.6 - Интерфейс коммуникации

```
#ifndef COMMUNICATION_H
#define COMMUNICATION_H
#include <stdint.h>

// Инициализация коммуникации (например, последовательного порта)
void comm_init();

// Обработка входящих команд
void comm_process();

// Разбор и выполнение команды
void comm_parse_command(uint8_t* buffer, uint8_t length);

// Отправка значения параметра
void comm_send_param(uint8_t param_id);

// Отправка всех параметров
void comm_send_all_params();

// Отправка истории параметра
void comm_send_history(uint8_t param_id);

#endif // COMMUNICATION_H
```

Листинг В.7 - Общие определения UI

```
#ifndef UI_H
#define UI_H

#include <stdint.h>

// Типы экранов
typedef enum {
      SCREEN_MAIN,
      SCREEN_EDIT,
      SCREEN_CHART,
      SCREEN_ERROR
} ScreenType;

// Глобальные переменные
extern int edit_param_id;
extern float edit_value;

// Функции для управления UI
void handle_confirmation(bool save);

#endif // UI_H
```

Листинг В.8 - Настройка LVGL

```
#ifndef LVGL_SETUP_H
#define LVGL_SETUP_H

#include "lvgl.h"

// Размер экрана (должен совпадать с config.h)
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240

// Инициализация LVGL и дисплея
void lvgl_init();
```

Окончание листинга В.8

```
// Обработчик тиков LVGL
void lvgl_tick_handler(void* param);

// Увеличение тика LVGL
void lvgl_tick_inc(int period_ms);

#endif // LVGL_SETUP_H
```

Листинг В.9 - Определение главного экрана

```
#ifndef MAIN_SCREEN_H
#define MAIN_SCREEN_H

#include <stdint.h>

// Создание главного экрана
void create_main_screen();

// Обновление главного экрана
void update_main_screen();

// Выбор параметра (direction: +1 или -1)
void select_parameter(int direction);

// Получение выбранного параметра
uint8_t get_selected_parameter();

#endif // MAIN_SCREEN_H
```

Листинг В.10 - Определение экрана редактирования

```
#ifndef EDIT_SCREEN_H
#define EDIT_SCREEN_H

// Создание содержимого экрана редактирования для параметра param_id
void create_edit_screen_content(int param_id);
```

Окончание листинга В.10

```
// Обновление экрана редактирования
void update_edit_screen();

// Корректировка значения параметра (direction: +1 или -1)
void adjust_parameter_value(int direction);

#endif // EDIT_SCREEN_H
```

Листинг В.11 - Определение экрана графиков

```
#ifndef CHART_SCREEN_H
#define CHART_SCREEN_H

// Создание экрана графика для параметра param_id
void create_chart_screen(int param_id);

// Обновление экрана графика
void update_chart_screen();

// Масштабирование осей графика
void chart_scale_x(int direction);
void chart_scale_y(int direction);

#endif // CHART_SCREEN_H
```