Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Работа допущена к защите

Директор ВШ КТиИС

\_\_\_\_\_\_\_\_\_\_\_ В.А. Сушников

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

РАБОТА БАКАЛАВРА

**Разработка программного обеспечения для системы управления стендом испытательным гидробарическим**

|  |  |  |
| --- | --- | --- |
| по направлению подготовки | |  |
| 09.03.01 Информатика и вычислительная техника | | |
| код и наименование | | |
| направленность (профиль) |  | |
| 09.03.01\_01 Разработка компьютерных систем | | |
| код и наименование | | |

Выполнил

студент гр.5130901/10101 <*подпись*> Д.Л. Симоновский

Руководитель

должность,  
ученая степень, ученое звание <*подпись*> А.А. Лавров

Консультант

по нормоконтролю <*подпись*> А.Г. Новопашенный

Санкт-Петербург

2025

Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

УТВЕРЖДАЮ

Директор ВШ КТиИС \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Сушников

«\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы**

студенту Симоновскому Даниилу Леонидовичу, группа 5130901/10101

фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: «Разработка программного обеспечения для системы управления стендом испытательным гидробарическим»

2. Срок сдачи студентом законченной работы: 21.05.2025

3. Исходные данные по работе: Техническое задание на разработку программного обеспечения стенда испытательного гидробарического; примеры реализаций различных проектов на аналогичной аппаратной базе; научная литература по теме автоматизации промышленных стендов.

4. Содержание работы (перечень подлежащих разработке вопросов): Изучение существующих аналогов разрабатываемого стенда для тестирования устройств под воздействием внешнего давления. Описание аппаратных средств, используемых для управления испытательным стендом. Описание алгоритмов управления испытательным стендом. Разработка программно-алгоритмических средств, поддерживающих одновременное управление и со штатной панели оператора, и с дублирующего интерфейса пользователя, выполненного на ПК, входящего в состав испытательного стенда. Разработка интерфейса пользователя для визуализации полученных данных в ходе испытания для ПК вышестоящих организаций.

5. Перечень графического материала (с указанием обязательных чертежей): не предусмотрен.

6. Перечень используемых информационных технологий, в том числе программное обеспечение, облачные сервисы, базы данных и прочие сквозные цифровые технологии (при наличии): Owen Logic, PyCharm, Конфигуратор СП300.

7. Консультанты по работе: консультант по нормоконтролю – А. Г. Новопашенный

8. Дата выдачи задания 21.04.2025

Руководитель ВКР А.А. Лавров

(подпись) инициалы, фамилия

Задание принял к исполнению 21.04.2025

(дата)

Студент Д.Л. Симоновский

(подпись) инициалы, фамилия

**РЕФЕРАТ**

На 60 с., 12 рисунков, 5 таблиц, 6 приложений

КЛЮЧЕВЫЕ СЛОВА: ФАЗЗИНГ, ПОИСК УЯЗВИМОСТЕЙ, ЯДРО LINUX, АППАРАТНАЯ ВИРТУАЛИЗАЦИЯ, СЕТЕВОЙ СТЕК

Тема выпускной квалификационной работы: «Разработка программного обеспечения для системы управления стендом испытательным гидробарическим».

**ABSTRACT**

60 pages, 12 figures, 5 tables, 6 appendices

KEYWORDS: FUZZING, VULNERABILITY SCANNING, LINUX KERNEL, HARDWARE VIRTUALIZATION, NETWORK STACK

The topic of the graduate qualification work is « Development of Software for the Control System of a Hydrobaric Test Bench».

**СОДЕРЖАНИЕ**

[СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ 8](#_Toc199163081)

[ВВЕДЕНИЕ 9](#_Toc199163082)

[1. ОБЗОР ИСПЫТАТЕЛЬНОГО СТЕНДА 11](#_Toc199163083)

[2. ОБЗОР АППАРАТНЫХ МОДУЛЕЙ УСТРОЙСТВА СИГ 13](#_Toc199163084)

[2.1. Программируемое реле 13](#_Toc199163085)

[2.2. Дисплей СП310-Б 15](#_Toc199163086)

[2.3. Частотный преобразователь ПЧВ1 17](#_Toc199163087)

[2.4. Датчики давления ПД100 18](#_Toc199163088)

[2.5. Электропривод ЭПК24АВ 19](#_Toc199163089)

[2.6. Выводы по разделу 21](#_Toc199163090)

[3. ОБЗОР РЕАЛИЗУЕМЫХ АЛГОРИТМОВ 21](#_Toc199163091)

[3.1. Общий алгоритм работы с СИГ 21](#_Toc199163092)

[3.2. Ручной режим 22](#_Toc199163093)

[3.3. Циклический режим 24](#_Toc199163094)

[3.4. Статический режим 27](#_Toc199163095)

[3.5. Выводы по разделу 29](#_Toc199163096)

[4. РАЗРАБОТКА ПРОГРАММЫ ДЛЯ СИГ 29](#_Toc199163097)

[4.1. Подключение периферии 29](#_Toc199163098)

[4.2. Разработка программы для СП310 39](#_Toc199163099)

[4.3. Разработка программы для компьютера 46](#_Toc199163100)

[4.4. Разработка программы для ПР200 51](#_Toc199163101)

[4.5. Разработка приложения для отображения графиков после испытаний 65](#_Toc199163102)

[4.6. Выводы по разделу 65](#_Toc199163103)

[5. ТЕСТИРОВАНИЕ УСТАНОВКИ СИГ 65](#_Toc199163104)

[5.1. Тестирование 65](#_Toc199163105)

[5.2. Выводы по разделу 65](#_Toc199163106)

[ЗАКЛЮЧЕНИЕ 66](#_Toc199163107)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 67](#_Toc199163108)

[Приложение 1 69](#_Toc199163109)

[Приложение 2 69](#_Toc199163110)

[Приложение 3 73](#_Toc199163111)

[Приложение 4 95](#_Toc199163112)

[Приложение 5 97](#_Toc199163113)

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

СИГ - стенд испытательный гидробарический.

Токовая петля - способ передачи информации с помощью измеряемых значений силы электрического тока.

RS-485 (Recommended Standard 485) - стандарт физического уровня для асинхронного интерфейса.

Modbus — открытый коммуникационный протокол, основанный на архитектуре ведущий — ведомый.

АЦП — аналогово‑цифровой преобразователь (ADC, Analog‑to‑Digital Converter) — устройство, которое преобразует непрерывный аналоговый сигнал (напряжение, ток) во взаимно‑однозначный цифровой код.

ЦАП — цифро‑аналоговый преобразователь (DAC, Digital‑to‑Analog Converter) — устройство, которое выполняет взаимно‑однозначное преобразование в непрерывный аналоговый сигнал заданной амплитуды входного дискретного цифрового кода.

ЖКИ-дисплей — жидкокристаллический дисплей — экран, в основе которого лежат жидкие кристаллы, которые используются для формирования заданного изображения/текста.

ВВЕДЕНИЕ

Разработка и создание испытательных стендов для тестирования оборудования под воздействием высокого внешнего давления является важной и востребованной задачей в современной промышленности. Подобные испытания обеспечивают возможность оценки надежности, долговечности и безопасности различных устройств, например водолазных баллонов, которые применяются в условиях, сопряженных с повышенным уровнем риска. Для проведения таких испытаний требуется использование специализированных стендов.

Компания АО "НПО "Прибор" [5] испытывает значительные затруднения в организации испытаний оборудования под высоким внешним давлением в условиях существующих решений. На территории Санкт-Петербурга отсутствуют компании, предоставляющие услуги, полностью соответствующие требованиям предприятия. Действующие аналоги, такие как ЦНИИ "Электроприбор" [17], оснащены испытательными камерами, объем которых значительно превышает потребности компании, что отрицательно сказывается на времени, затрачиваемом на проведения тестов, а таже их географическая удаленность от Санкт-Петербурга влечет за собой сложности с транспортировкой испытываемого оборудования. Существуют также компании, производящие подобные испытательные стенды, например компания Hydrofab.ru [2]. На их сайте указано, что у них есть стенды с автоматизированными решениями, однако они не предоставляют примеров их практической реализации. Учитывая длительность и трудоемкость согласования требований с внешними подрядчиками, наиболее целесообразным представляется создание собственного испытательного стенда на базе предприятия. Кроме того, внутренняя разработка обеспечит возможность гибкой модернизации системы, как в процессе разработки, так и в будущем, например добавление новых режимов или изменение конструкции.

Разработка стенда испытательного гидробарического (далее СИГ) с автоматизированным управлением направлена на удовлетворение производственных потребностей АО "НПО "Прибор" и оптимизацию процесса испытаний. Основная цель данной работы заключается в создании программного обеспечения, обеспечивающего функционирование СИГ в полуавтоматическом режиме. Это включает реализацию ручного и автоматического режимов управления, а также визуализацию и демонстрацию результатов испытаний. При этом конструкция стенда рассматривается как уже реализованная, что позволяет сосредоточиться на создании алгоритмов управления и интеграции с используемыми компонентами.

# ОБЗОР ИСПЫТАТЕЛЬНОГО СТЕНДА

Прежде всего рассмотрим, что такое СИГ. Это установка, для создания внешнего давления на испытываемое устройство (сосуд). Создаваемое давление может достигать 38 МПа. Испытуемое устройство помещается внутрь герметичного цилиндрического пространства установки объемом 1 м3 после чего оно заполняется водой, герметично закрывается и начинается процесс испытания.

Для проведения испытаний предусмотрены три режима работы установки:

1. **Ручной**. Данный режим, как понятно из названия полностью управляется человеком. Пользователю будет необходимо самостоятельно задавать процент работы насоса, тем самым регулируя скорость набора давления. Данный режим служит в первую очередь для тестирования самой установки, а также будет использоваться при сборе данных для построения модели, однако также останется в итоговой версии для диагностических целей или проведения испытаний, которые были не предусмотрены на стадии разработки. Подробнее этот режим будет рассмотрен в разделе 3.2.
2. **Циклический**. Это первый из предусмотренных автоматических режимов. Он проверяет испытываемое устройство на работоспособность при многократном наборе и сбросе давления. Для этого автоматически с заданной скоростью СИГ должен нагнетать давление в баке, после чего сбрасывать его, путем открытия клапана. Более подробно данный режим будет рассмотрен в соответствующем разделе 3.3.
3. **Статический**. Это второй автоматический режим, в нем реализуется возможность ступенчатого повышения давления до заданного значения с удержанием на каждой ступени в течение определенного времени при заданной скорости набора давления. Таким образом испытываемое устройство проверяется на устойчивость к длительному нахождению под определенным давлением. Данный режим будет подробно рассмотрен в разделе 3.4.

Автоматизация управления позволит выполнять работу с минимальным участием оператора, что существенно повысит эффективность испытательного процесса, а также данный подход позволит обеспечить большую безопасность самого оператора в ходе эксплуатации установки.

В рамках работы предполагается использование следующих компонентов:

* Программируемое реле ПР200 [13] — контроллер, отвечающий за реализацию логики работы стенда, а также опрос всех остальных устройств и датчиков.
* Панель оператора СП310-Б [15] — экран оператора, предоставляющий возможность отслеживания процесса работы системы, ввода параметров режимов, а также запуск и остановку их работы.
* Удаленное рабочее место оператора для дублирования СП310-Б — место, вынесенное отдельно от основной установки, на базе компьютера со специально разработанной программой, которая дублирует СП310-Б. Используется с целью вынесения рабочего места оператора дальше от места проведения испытаний с целью обеспечения безопасности пользователя.
* Частотный преобразователь ПЧВ1 [14] — устройство для управления плунжерным насосом НД 25/400 К14А [12], который обеспечивает набор давления в системе.
* Датчики давления ПД100‑ДИ60.0‑XX‑0.5 [11] — применяются для получения текущих параметров давления в системе.
* Электропривод ЭПК24АВ [18] – электропривод для вращения краном сброса давления.

Результаты испытаний будут сохраняться в формате, удобном для дальнейшей обработки и анализа, который будет проводиться с помощью программного обеспечения, предоставляющего заказчику графическую интерпретацию данных.

Структурная схема устройства представлена на рис.1.1.

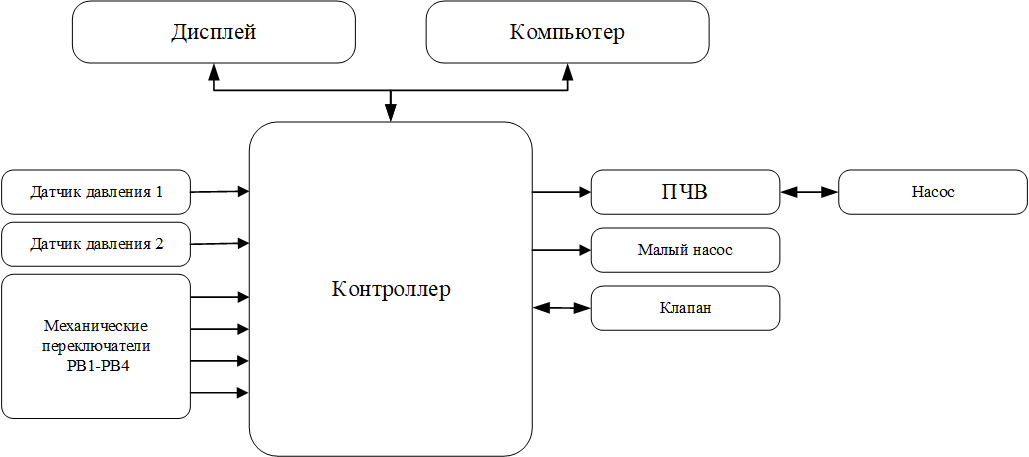


Рис.1.1. Структурная схема устройства СИГ.

Контроллером будет выступать ПР200-220.4.2.0, его выбор обусловлен многими преимуществами, которые делают его удобным в рамках данного проекта:

Во-первых, оборудование компании ОВЕН широко доступно на российском рынке. ОВЕН — российская компания, занимающаяся разработкой и производством средств автоматизации. Это особенно важно в условиях, когда импорт затруднён.

Во-вторых, оборудованием данной компании пользуется большое количество специалистов, благодаря чему, существует множество примеров работы, готовых решений и инструкций на русском языке, которые можно использовать в ходе разработки.

Помимо вышеперечисленных плюсов компания предлагает широкий выбор других устройств, помимо контроллеров, таких как панель оператора СП310, датчики давления ПД100 и преобразователи частоты ПЧВ1. Эти устройства имеют стандартные интерфейсы для взаимодействия между собой, что упрощает процесс настройки, а использование оборудования одного производителя позволяет избежать трудностей с подключением их друг к другу, что экономит время при настройке.

# ОБЗОР АППАРАТНЫХ МОДУЛЕЙ УСТРОЙСТВА СИГ

## Программируемое реле

ПР200‑220.4.2.0 является одной из модификаций программируемого реле серии ПР200 (функциональная схема приведена на рис.2.1) производства компании ОВЕН и предназначено для решения локальных задач промышленной автоматизации [13].

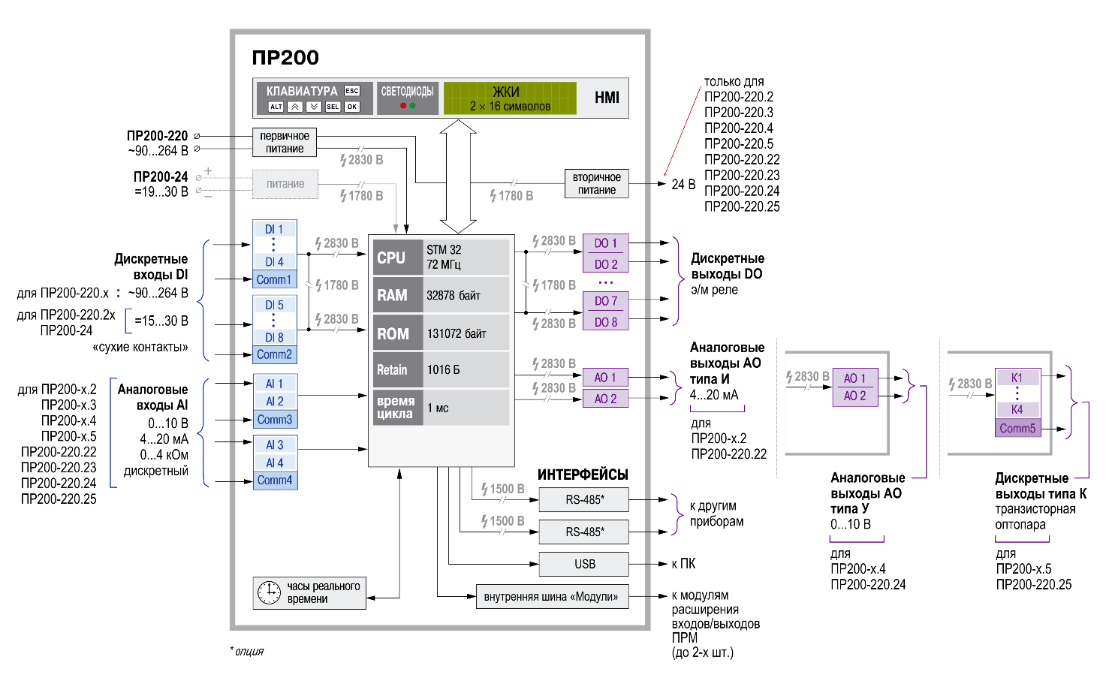


Рис.2.1. Функциональная схема ПР200.

Ниже приведены основные параметры ПР200‑220.4.2.0 [13]:

* Питание прибора: переменное напряжение 90 - 264 В (номинальное 230 В, 50 Гц) или постоянное 127 - 373 В.
* Встроенный источник питания (ВИП): 24 В DC, 100 мА (для питания аналоговых датчиков).
* Дискретные входы: 8 каналов для фазового входного сигнала ~230 В AC.
* Аналоговые входы: до 4 каналов, измерение сигналов 0 - 10 В, 4 - 20 мА или сопротивления до 4 кОм с 12‑бит АЦП и периодом опроса ≤ 10 мс.
* Дискретные выходы: 8 каналов (электромеханические реле), нагрузочная способность реле: до 5 А при 250 В AC и до 3 А при 30 В DC; гальваническая развязка по двум выходам в группе.
* Аналоговые выходы**:** 2 канала ЦАП «параметр‑напряжение», 0 - 10 В.
* Интерфейсы связи: два независимых порта RS‑485 с поддержкой протоколов Modbus RTU и Modbus ASCII в режимах Master/Slave.

В приборе установлен микроконтроллер с 128 КБ ПЗУ и 32 КБ ОЗУ, динамическим стеком и 1016 байтами Retain‑памяти для хранения постоянных переменных; минимальное время цикла выполнения программы — 1 мс. Программирование реализуется посредством среды Owen Logic на языке функциональных блок‑диаграмм (FBD), загрузка алгоритма осуществляется через встроенный порт miniUSB [13].

Дискретные входы объединены в две группы по четыре канала с групповой гальванической развязкой (изоляция 2830 В, групповая — 1780 В). Аналоговые входы не имеют изоляции и поддерживают работу в дискретном режиме [13].

Дискретные выходы выполнены, используя электромеханические реле, которые обеспечивают надёжную коммутацию различных нагрузок; аналоговые выходы реализованы при помощи цифро‑аналогового преобразователя (ЦАП), гарантирующего высокую точность формирования выходного напряжения [13].

Два интерфейса RS‑485 позволяют интегрировать прибор в верхний уровень автоматизации, организовать опрос датчиков и управление исполнительными устройствами по стандартным протоколам Modbus [13].

В устройстве присутствует символьный ЖКИ‑дисплей 2×16 символов с подсветкой, который поддерживает кириллицу и латиницу, а также содержит шесть механических кнопок для управления им. [13].

Для увеличения числа входов/выходов допускается подключение до двух модулей расширения ПРМ по внутренней шине, каждый из которых имеет собственное питание и отличается набором дискретных и аналоговых каналов [13].

## Дисплей СП310-Б

Панель оператора СП310‑Б (внешний вид приведен на рис.2.4) является частью серии сенсорных панелей оператора ОВЕН СП3хх (диагонали 7″/10,1″/15,6″) которая предназначена для отображения различных данных, управления и ведения архива событий и значений. Конфигурирование осуществляется в среде «Конфигуратор СП300» [15].

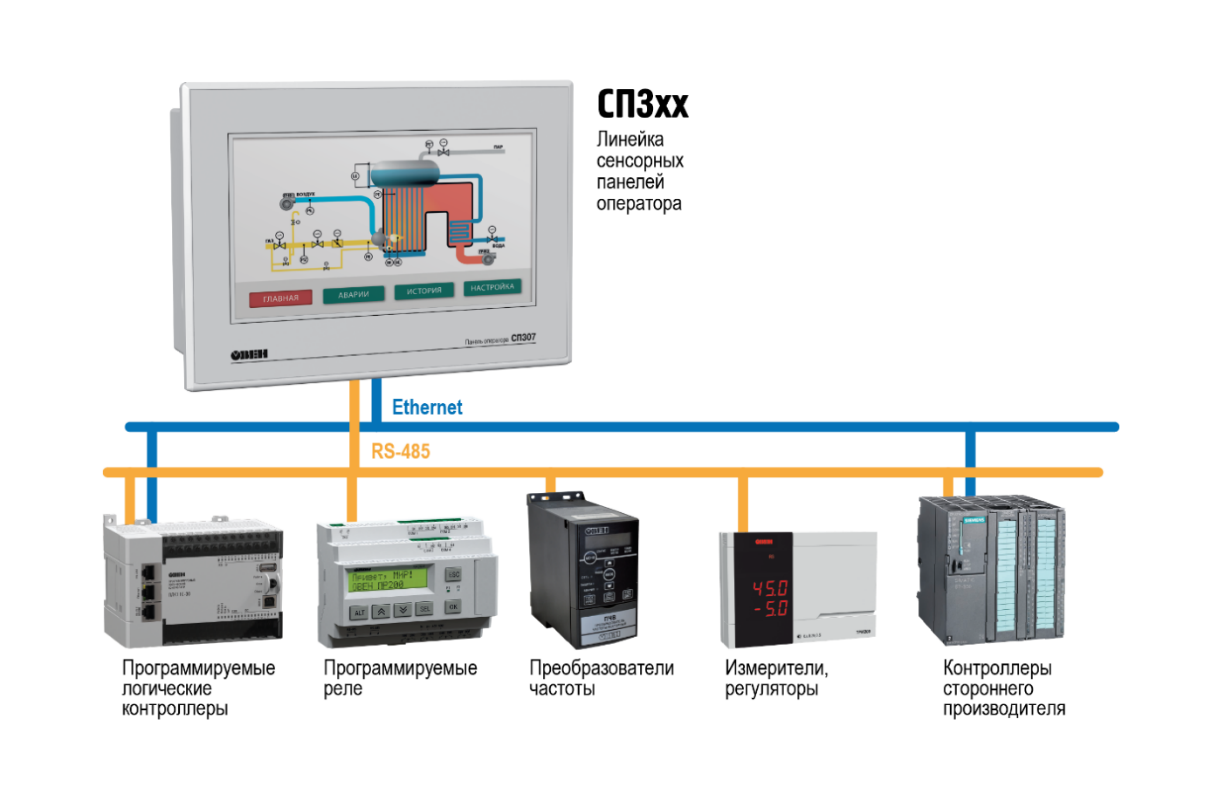


Рис.2.4. Примеры использования СП3xx.

В основе панели оператора лежит 32‑битный ARM‑микроконтроллер AT91SAM9G35‑CU с тактовой частотой 400 МГц. Объём встроенной Flash‑памяти составляет 128 МБ (минимум 75 000 циклов перезаписи), а оперативной 128 МБ, также имеется энергонезависимый RTC питающийся от элемента CR2032 (±0,7 с/сутки) [15].

В качестве самого дисплея используется TFT‑LCD диагональю 10,1″ с LED‑подсветкой, 16,7 млн цветов (TrueColor) и разрешением 1366×768 px. Яркость 250 кд/м² и контрастность 500:1, что гарантирует чёткое изображение; ресурс подсветки — не менее 50 000 ч при 25 °C, яркость регулируется программно [15].

Для организации управления в панели оператора СП310 существует два универсальных COM‑интерфейса RS‑232/RS‑485 (Download‑ и PLC‑порт), они поддерживают Modbus RTU в режимах ведущего и ведомого, а также Modbus ASCII, только в режиме ведущего [15].

Прибор питается постоянным напряжением 23 - 27 В (номинал 24 В). Максимальный потребляемый ток — 0,27 А, мощность — до 10 Вт (при старте пусковой ток может превышать номинальное значение в 10 раз до 25 мс, что требует блока питания мощностью не менее 30 Вт) [15].

## Частотный преобразователь ПЧВ1

Частотный преобразователь ПЧВ1 (внешний вид приведен на рис.2.5) представляет собой универсальное устройство для управления асинхронными двигателями в системах промышленной автоматизации [14].



Рис.2.5. Внешний вид ПЧВ1.

Прибор позволяет управлять оборотами подключенного к нему насоса, тем самым давая возможность изменять скорость набора давления в СИГ.

Для управления в ПЧВ1 предусмотрен внутренний ПЛК, однако в рамках данного проекта удобнее воспользоваться управлением по средствам RS-485 Modbus RTU. Для этого выполним настройку протокола используя встроенный дисплей (он расположен на передней панели ПЧВ1, см. рис.2.5) [14].

Для добавления ПЧВ1 в общую систему управления СИГ через шину RS‑485 в режиме ведомого по протоколу Modbus RTU необходимо задать параметры, приведенные в табл.2.1.

Таблица .

Сетевые параметры в ПЧВ1

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | Адрес (hex) | Значение | Обоснование |
| F12.00 – Режим «Ведущий/Ведомый» | 0x0C00 | 0 | Режим «Slave» (ведомый) для передачи управления центральным контроллером, исключает конфликт ведущих. |
| F12.01 – Адрес устройства в сети Modbus | 0x0C01 | 1 |  |
| F12.02 – Скорость передачи данных | 0x0C02 | 5 | 38400 бит/с – максимальная допустимая скорость, что позволяет быстро передавать данные между ведущим и ведомыми. |
| F12.03 – Формат данных (четность, биты, стоп‑биты) | 0x0C03 | 1 | (E, 8, 1) – проверка по чётности даёт дополнительный уровень обнаружения ошибок. |

Выбранные параметры (чётность чет, 8 бит посылка и 1 стоп бит) являются наиболее распространёнными в промышленных сетях, они минимизируют риск ошибок при работе по протоколу в условиях помех [3].

После настройки ПЧВ1 можно управлять по протоколу ModBus, для этого в частотном преобразователе существует несколько регистров, в данном проекте будет использоваться только 2 из них, один для задания частоты, а второй для включения и выключения двигателя. Их подробное описание приведено в Приложение .

## Датчики давления ПД100

ПД100‑ДИ60.0‑XX‑0.5 — это промышленный измерительный преобразователь избыточного давления серии ПД100 (его внешний вид представлен на рис.2.6) производства ОВЕН [11].



Рис. 2.1. Внешний вид ПД100.

Прибор оснащён силовым сенсором КНК с мембраной из нержавеющей стали, что обеспечивает высокую химическую стойкость при контакте с газами, паром, водой и слабоагрессивными средами. За счёт микропроцессорного нормирования сигнала преобразователь гарантирует стабильность нулевой точки и минимизацию дрейфа параметров в широком диапазоне температур и вибраций [11].

Для модификации ПД100‑ДИ60.0‑XX‑0.5 верхний предел измерений составляет 60,0 МПа (нижний предел — 0 МПа) [11], что позволяет решать задачи контроля как низких, так и сверхвысоких давлений на промышленных установках. Такой диапазон позволяет с запасом удовлетворить потребности установки СИГ. Класс точности модификации ±0,5 % от полного диапазона [11], что обеспечивает высокую воспроизводимость результатов измерений.

Выходной сигнал — унифицированный токовый 4 - 20 мА по двухпроводной схеме (токовая петля).

Таким образом ПД100 является хорошим выбором в условиях данного испытательного стенда т.к. токовая петля обеспечивает надежную передачу данных, диапазон соответствует требованиям установки, а погрешность измерений 0.5% позволяет получить результат, достаточный для последующего анализа.

## Электропривод ЭПК24АВ

Электропривод ЭПК24АВ представляет собой устройство (внешний вид устройства представлен на рис.2.7.), разработанное для автоматизированного управления смесительными клапанами в системах автоматизации [10].



Рис.2.7. Внешний вид ЭПК24.

Данный привод предназначен для регулирования положения 2‑ и 3‑ходовых смесительных клапанов с крутящим моментом до 10 Н·м. Управление осуществляется по аналоговому вольтовому сигналу 010 В. Предусмотрена линия обратной связи, аналогично 010 В [10].

Диапазон поворота штока привода—90°, время максимального поворота составляет 35 с. Привод обладает возможностью ручного вмешательства посредством рукоятки и кнопки разблокировки. Монтаж на клапан осуществляется при помощи сменных адаптеров, поставляемых в комплекте [10].

Привод рассчитан на универсальное питание 24 В ± 10 %. Его максимальная потребляемая мощность в процессе работы не превышает 5 Вт. [10].

Этот привод хорошо подходит для контроля клапана слива, где не нужна большая скорость поворота, а важна точность для поддержания определенной скорость сброса давления. Интерфейс 010 В не является лучшим выбором в условиях работы с трехфазным двигателем и ПЧВ по причине электромагнитных помех, однако его подключение осуществляется по средствам кабеля с комбинированным экраном (фольга + оплётка), что позволяет добиться практически полной защиты от электромагнитных помех, а в свою очередь применение такого интерфейса упрощает отладку и подключение к ПР200.

## Выводы по разделу

В ходе данного раздела были рассмотрены основные устройства управления и измерения в СИГ, а также рассмотрены их функциональные особенности.

В проекте используется множество интерфейсов для взаимодействия между устройствами, а именно RS-485 Modbus RTU, токовая петля и сигнал 0 – 10 вольт.

В рамках этого раздела не был рассмотрен один ключевой элемент управления – а именно удаленно рабочее место оператора т.к. это может быть любое устройство, удовлетворяющее минимальным требованиям программы управления, поэтому рассматривать его технические особенности нет необходимости.

# ОБЗОР РЕАЛИЗУЕМЫХ АЛГОРИТМОВ

## Общий алгоритм работы с СИГ

Прежде чем перейти к рассмотрению конкретных алгоритмов работы, необходимо рассмотреть общий принцип работы с СИГ. Для начала рассмотрим пневмогидравлическую схему устройства на рис.3.1.

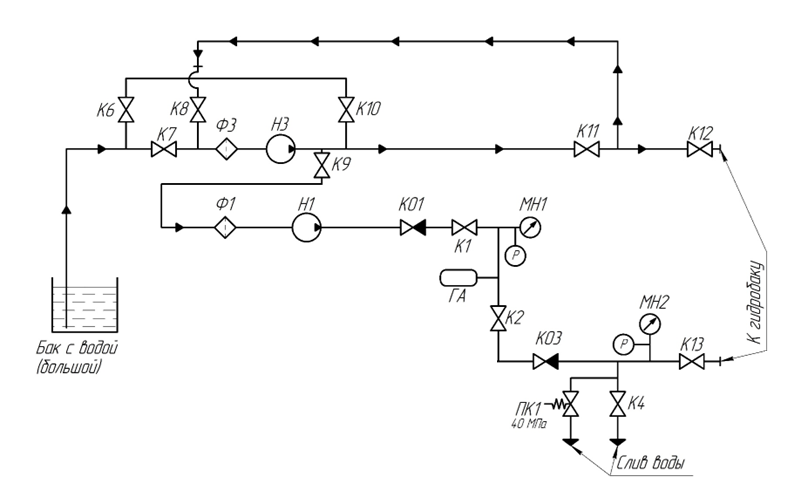


Рис.3.1. Пневмогидравлическая схема СИГ.

На рис.3.1 изображен способ соединения клапанов, насосов и манометров (датчиков давления). Перед использованием какого-то режима, описанного в следующих пунктах, необходимо подготовить СИГ к испытаниям.

Перед началом испытаний в гидробак загружается испытываемое устройство, после чего он сверху закрывается крышкой. Наверху крышки находится дополнительный ручной клапан для стравления воздуха при наборе воды.

После того, как испытываемое устройство помещено внутрь, необходимо заполнить гидробак водой, для этого требуется настроить гидравлическую линию, по указанной схеме на рис.3.1 требуется перекрыть краны К4, К6, К8, К9, К10, К13. Краны К7, К11, К12 при заполнении гидробака должны быть открытыми. После настройки клапанов необходимо запустить насос Н3, это можно сделать либо с использованием панели оператора СП310 в любом из режимов работы, либо из программы с удаленно рабочего места оператора, аналогично в окне любого из режимов, либо первой кнопкой, расположенной на щите.

После набора воды (когда из клапана для стравливания кислорода пойдет вода) насос Н3 отключается либо повторным нажатием кнопки на щите, либо выключением соответствующей кнопки в любом режиме, как на СП310, так и на удаленном рабочем месте оператора.

После заполнения необходимо подготовить стенд к работе по режиму, для этого необходимо закрыть К11 и К12, открыть К1, К2 и К13. После чего стенд можно использовать по любому из режимов работы.

Для корректной работы необходимо держать Н3 включенным, параллельно с Н1, что видно по схеме т. к. Н3 подает воду в Н1.

После завершения испытаний давление стравливается путем открытия пары К4 вплоть до 0 по датчикам давления, после чего воду необходимо откачать, для этого необходимо закрыть К1, К2 и К13, открыть К12, К8, К6 и К10, после чего включить Н3, который откачает воду из гидробака обратно в бак с водой.

На этом испытание заканчивается.

## Ручной режим

По техническому заданию необходимо реализовать три режима работы, первый из которых – ручной. В данном режиме не предусматривается разработка какой-либо алгоритмической составляющей, он должен позволять управлять основным двигателем напрямую, без различных режимов работы.

Управление должно происходить как посредствам кнопок, так и используя экран СП310-Б.

С экрана должна быть возможность задавать любой процент работы двигателя от 25 до 100, включительно, а также 0. Данный диапазон обусловлен требованиями к эффективной работе двигателя. В случае попытки задания величины, отличной от диапазона необходимо действовать по алгоритму, приведенному на рис.3.2.

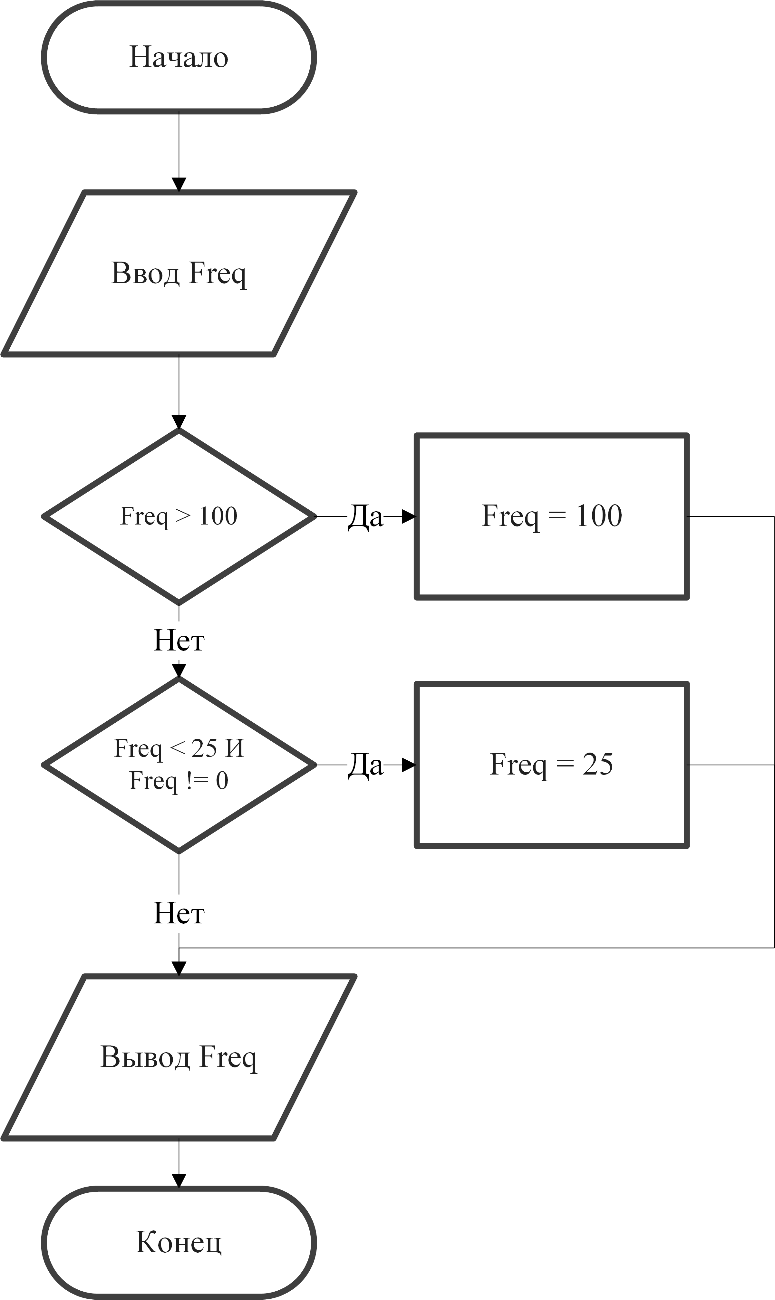


Рис.3.2. Алгоритм обработки  
вводимого значения частоты.

Помимо ввода значения с дисплея, должна быть возможность задания частоты с кнопок, подключенных к ПР. Для этого используются 2 кнопки, которые работают по одному алгоритму, с разницей лишь в задаваемом проценте работы ПЧВ. Для одной кнопки это значение 50%, а для второй 100%. Механический переключатель должен иметь 2 режима работы:

1. По зажатию дольше 1 секунды выставлять частоту (50% от максимума для первой кнопки и 100% для второй) без возможности сменить её из других источников, по отжатию выставлять 0%.
2. По однократному нажатию (длительностью менее 1 секунды) выставлять частоту (50% от максимума для первой кнопки и 100% для второй) с возможностью смены её из любого источника. Если частота не изменялась, то по повторному нажатию на механический переключатель частота должна выставляться 0%.

Таким образом ручной режим позволит управлять ПЧВ напрямую, что упростит первичную отладку стенда, а также предоставит возможность в ручном режиме производить испытания, отличные от заранее заготовленных режимов.

## Циклический режим

Следующим рассматриваемым режимом является циклический. Как и было сказано ранее, он служит для проверки испытываемого изделия на устойчивость к периодическому подъему и падению давления с заданной скоростью.

Запуск и остановка режима должны быть доступна как с сенсорного дисплея СП310 или удаленного рабочего места оператора, так и используя механические переключатели. Также на экране должна быть индикация о текущем статусе режима, а именно запущен он или остановлен.

В результате выполнения 1 цикла режима должен получиться график, приведенный на рис.3.3.

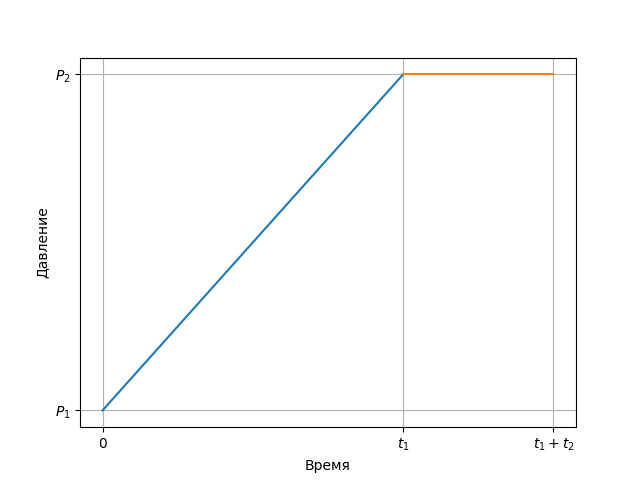


Рис.3.3. Один цикл в циклическом режиме.

Сначала идет набор давления от P1 до P2 за время t1, однако для испытания не важно само время набора, а важна скорость, именно её и будет задавать пользователь. Стоит также отметить, что давление P1 не всегда является нулевым, поскольку после первого цикла, сброс может происходить не полностью, а, например, до 5 МПа. Также важно отметить, что не всегда график будет таким «идеальным», в какой-то момент, например ближе к пиковому давлению, на большой требуемой скорости двигатель может не справиться с нею, в таком случае время t1 будет отличаться от «идеального» в большую сторону. Важно учитывать, что целью является именно удержание скорости, а не достижение заданного давления за «идеальное» время.

Далее, после достижения заданного давления, необходимо его удерживать заданное пользователем время, после чего сбросить давление до заданной границы, используя ЭПК24АВ и повторить цикл заданное количество раз. В последний цикл автоматический сброс выполнять необходимости нет.

В итоге для работы режима потребуется ввести 5 значений:

* Конечное давление — давление, которое показывает до какого значения необходимо выполнять набор.
* Скорость набора давления — это то значение скорости, к которому должна стремиться система во время набора конечное давления.
* Время удержания — то, сколько необходимо удерживать конечное давление до выключения режима.
* Количество повторов цикла — то, сколько необходимо раз набрать требуемое давление.
* Нижнее значение давление — то, до какого давления необходимо выполнять сброс.

Эти значения должны вводиться оператором на СП310 или с удаленно рабочего места оператора и проверяться на корректность.

После завершения удержания на последнем цикле на экране индикация должна показывать, что режим остановлен. В случае, если на этом испытания завершены, необходимо действовать по алгоритму, описанному в пункте 3.1.

По результату проведения испытания должен получиться график, приведенный на рис.3.4.

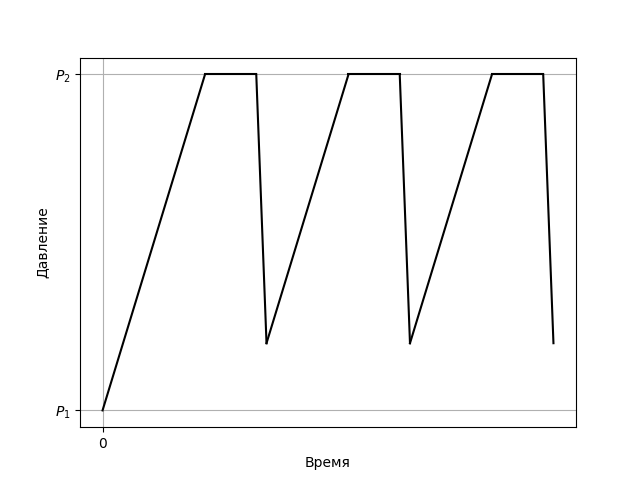


Рис.3.4. Несколько циклов в циклическом режиме.

В ходе выполнения работы не должно быть допустимо изменение параметров работы алгоритма без выключения работы системы.

Таким образом циклический режим позволяет проверить испытываемое изделие на устойчивость к многократному набору и сбросу заданного давление при заданной скорости, что позволит убедиться в его пригодности при многократном использовании под высоким давлением.

## Статический режим

Последним режимом, который необходимо реализовать, является статический. Он необходим для проверки изделия под длительным воздействием различных давлений, при этом подъемом на очередное давление происходит со скоростью, задаваемой пользователем.

Запуск и остановка режима должны быть доступна как с сенсорного дисплея СП310 или удаленного рабочего места оператора, так и используя механические переключатели. Также на экране должна быть индикация о текущем статусе режима, а именно запущен он или остановлен.

В результате выполнения режима должен получиться график, приведенный на рис.3.5.

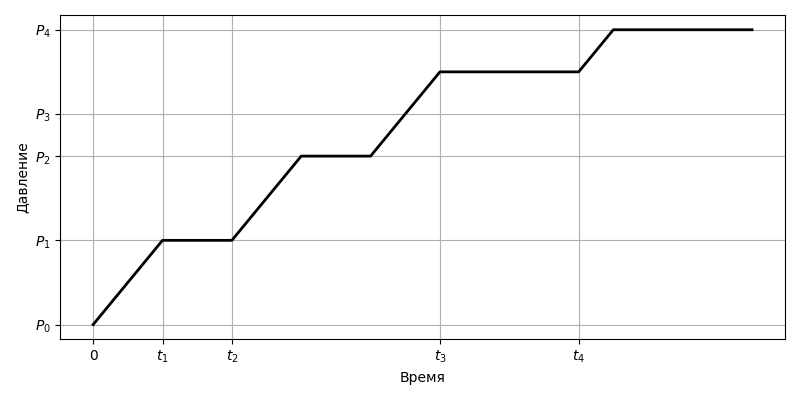


Рис.3.5. График – результат работы статического режима.

В первую очередь рассмотрим обозначения на оси ординат:

* Разница между P2 и P1 – величина «ступени», т.е. значение, на которое отличается предыдущее место удержания и следующее. Важно отметить, что эти значения высчитываются не относительно точки P0, а относительно 0. Т.е. первая точка P1 при значении P0 меньше величины шага будет всегда одинаковой и равной шагу. Если же окажется ситуация, когда P0 больше величины шага, необходимо точкой P1 выбрать ближайшую кратную значению ступени величину, большую, чем P0. Доступ к изменению значения ступени не должен предоставляться оператору по средствам дисплея или удаленного рабочего места оператора и стандартно должен быть равен 0.5 МПа, однако изменение должно быть возможно посредствам экрана непосредственно на ПР200.
* P3 – точка промежуточного давления. Относительно этой точки будет меняться время удержания давления. Это нужно потому, что чаще всего практически нет смысла по долгу удерживать устройство длительное время при малом значении давления, однако для протокола необходимо, поэтому в программе должно быть предусмотрена точка, относительно которой время удержания будет меняться.
* P4 – максимальное значение давления. Важно отметить, что оно не обязано быть кратно непосредственно шагу ступени, в таком случае очередной точкой удержания будет выбираться именно максимальное давление, а не очередное значение, которое должно быть при заданном шаге.

Далее перейдем к оси абсцисс:

* Время между началом набора и t1, как и все последующие времена набора ступени, задаются не через значение времени, а через скорость набора. Механизм полностью повторяет аналогичный для циклического режима в разделе 3.3.
* Разница между t2 и t1 – время удержания до промежуточного давления. В это время давление должно удерживаться на одном уровне.
* Разница между t4 и t3 – время удержания после промежуточного давления. В это время давление должно удерживаться на одном уровне.

Все значения помимо шага ступени должны вводиться оператором на СП310 или с удаленного рабочего места оператора и проверяться на корректность.

После завершения испытания на экране индикация показывает, что режим остановлен, после чего пользователь может ввести новые параметры, в случае необходимости продолжения испытаний, либо завершить их. В случае, если испытания завершены, необходимо действовать по алгоритму, описанному в 3.1.

В ходе выполнения режима параметры должны быть недоступны для редактирования.

Таким образом статический режим позволит проверить испытываемое устройство на устойчивость к длительному воздействию давления, что позволит убедиться в его пригодности при работе под давлением в течении заданного времени.

## Выводы по разделу

В ходе данного раздела был подробно описан алгоритм при подготовке к работе СИГ, а также алгоритм действий после завершения испытаний.

Также были рассмотрены три доступных режима работы СИГ, и подробно описаны оба автоматических режима работы. Были приведены необходимые для работы каждого из алгоритмов параметры, а также их смысл.

# РАЗРАБОТКА ПРОГРАММЫ ДЛЯ СИГ

## Подключение периферии

Перед переходом непосредственно к основной части главы, перейдем в Owen Logic и выполним создание проекта для ПР200. На рис.4.1 мы выбираем используемое устройство.

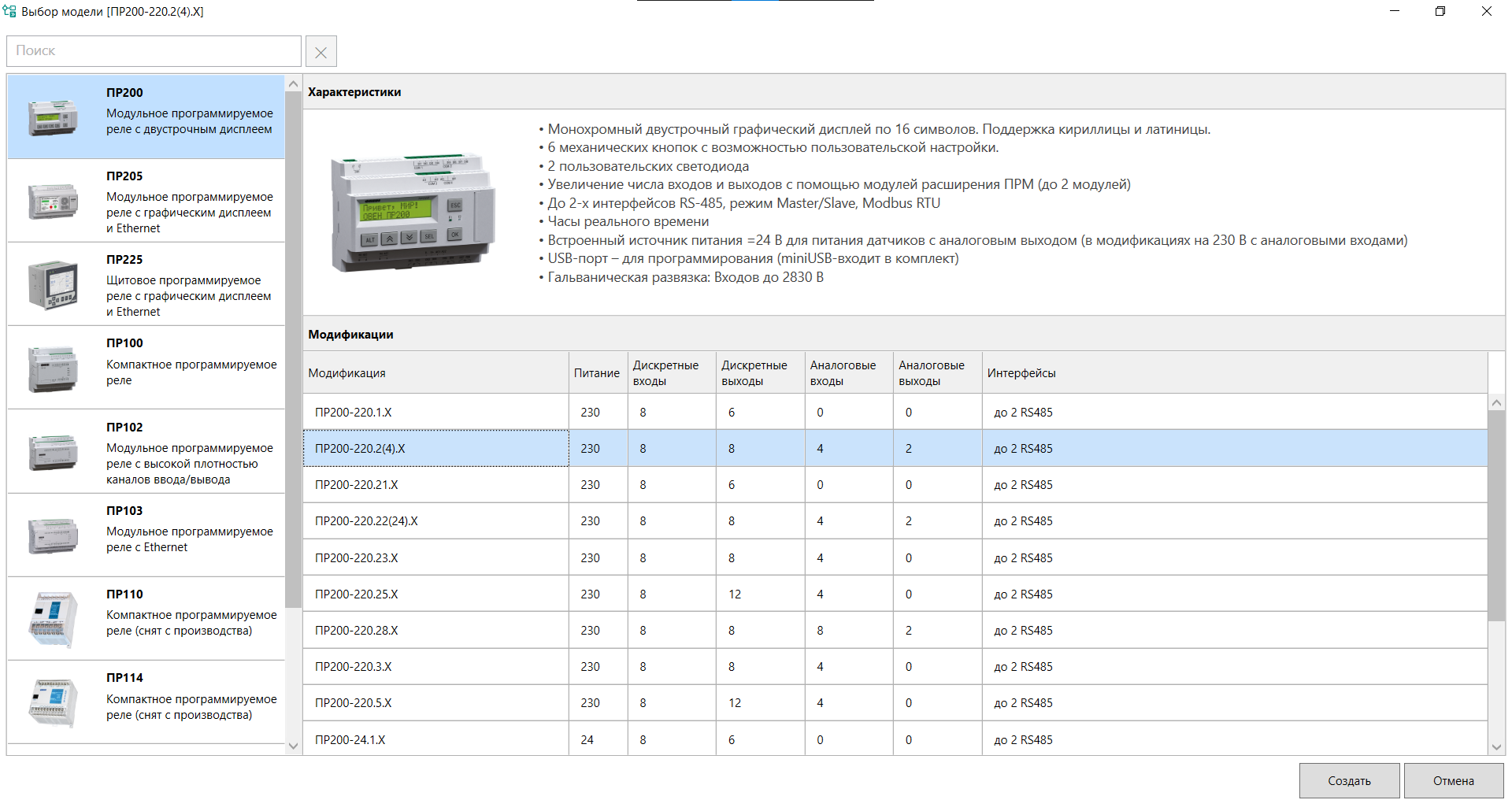


Рис.4.1. Выбор устройства ПР200 в среде Owen Logic.

После создания проекта появляется экран с непосредственно основным меню для разработки, оно приведено на рис.4.2. На нем мы видим переменные среди (на данный момент только служебные), а также папки с основными компонентами разработки (справа) и различные дополнительные элементы управления (сверху). В данной работе не будет подробного описания работы с данной средой, а лишь результаты моей работы в ней.

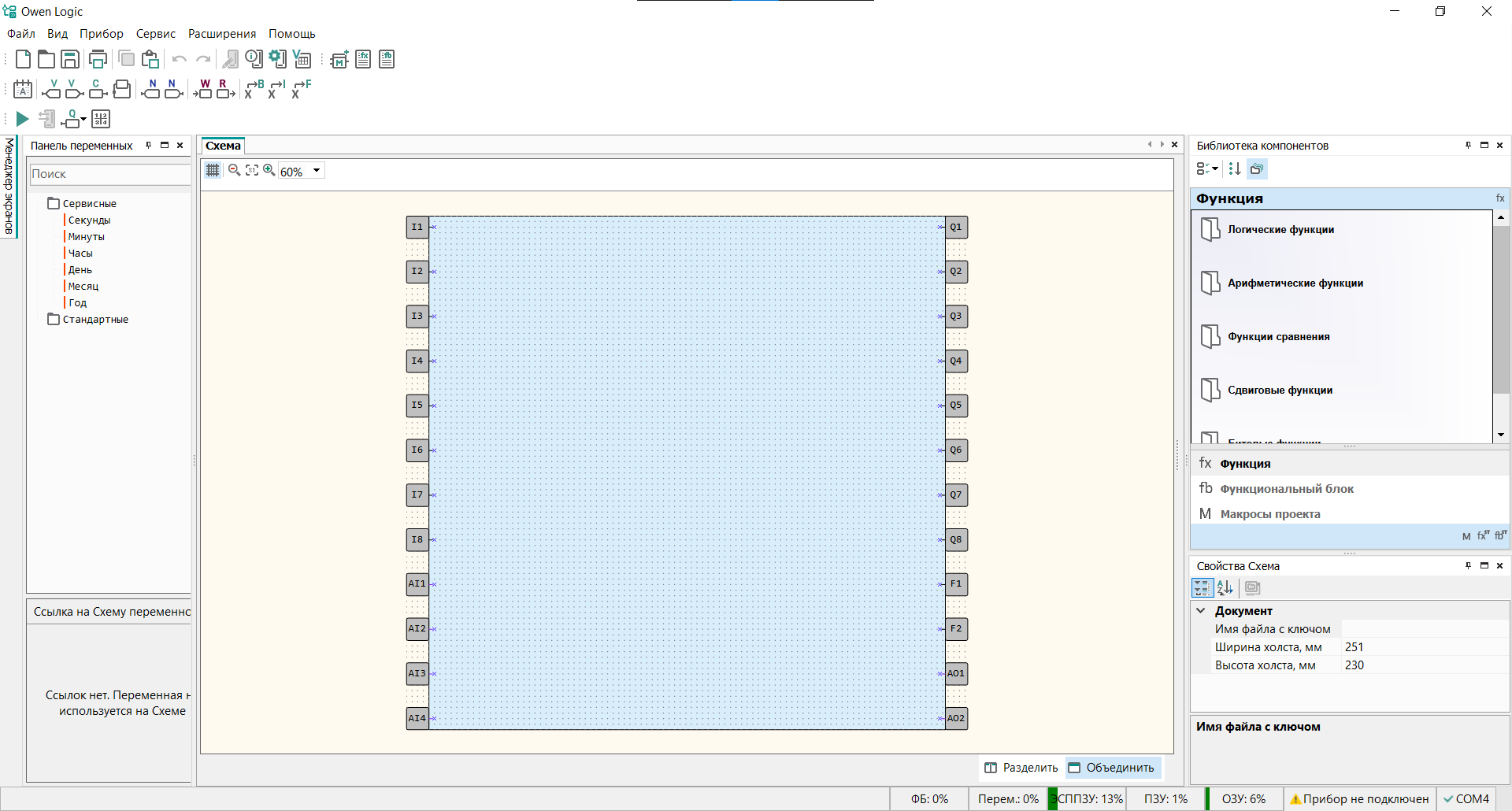


Рис.4.2. Устройство ПР200 в среде Owen Logic.

Прежде чем перейти непосредственно к разработке необходимо описать, каким образом отдельные элементы управления будут подключаться к контроллеру ПР200.

Рассмотрим подключение устройств к контроллеру на физическом уровне, оно показано на рис.4.3.

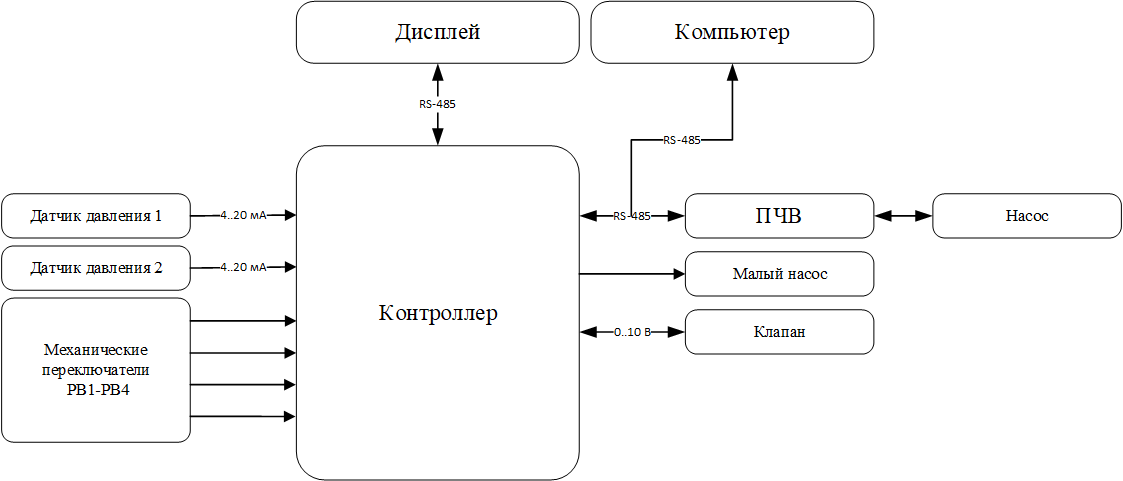


Рис.4.3. Схема подключения устройств к контроллеру.

Для управления механическими переключателями и включения малого насоса используются дискретные входы и выходы контроллера. Эти элементы подключаются следующим образом:

1. Кнопки: Каждая кнопка подключается к соответствующему дискретному входу контроллера через нормально разомкнутую контактную пару. Один контакт кнопки соединяется с общим проводом (GND), а второй — с входным каналом контроллера.
2. Малый насос: Включение насоса осуществляется через дискретный выход контроллера, который управляет реле. Выход контроллера соединяется с управляющим входом реле, обеспечивая замыкание силовой цепи насоса при подаче сигнала.

Подробнее схему подключения можно рассмотреть в [16], где приведена электрическая схема устройства.

В среде Owen Logic мы сохраним состояние кнопок в соответствующие переменные (как показано на рис.4.4) и будем использовать позднее. Важно отметить, что для кнопок 1 и 2 на входах дополнительно использовались триггеры переднего фронта, а для кнопок 3 и 4 - нет. Это связано с тем, что кнопки 1 и 2 используются для запуска и остановки большого и малого насоса, где время их удержания значения не имеет, а вот кнопки 3 и 4 используются в ручном режиме, где в зависимости от удержания будут разные механизмы работы (см. пункт 3.2)..

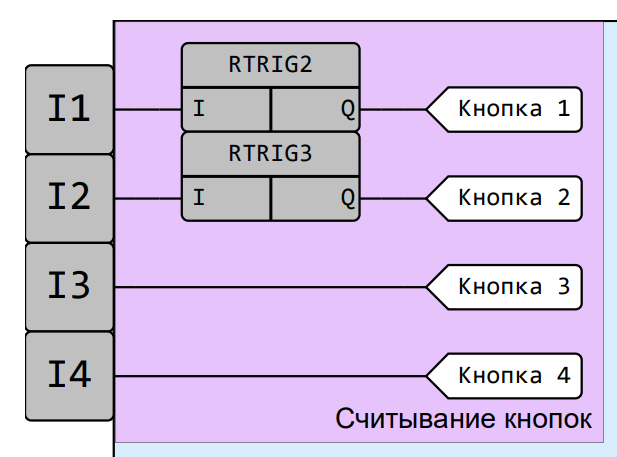


Рис.4.4. Считывание данных с кнопок.

С насосом тот же принцип, но переменная на этот раз создается для последующего управления. Эта переменная показана на рис.4.5.

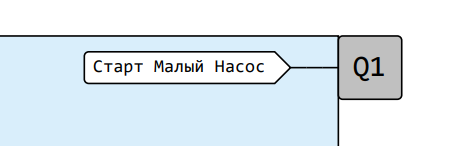


Рис. .. Вывод переменной для управления малым насосом.

Для подключения ПД100 к контроллеру используется токовая петля – распространенный способ подключения датчиков в промышленных системах автоматизации благодаря своей надежности и экономической эффективности, а также высокой точности передачи данных в условиях интенсивных электромагнитных помех и на значительные расстояния [7]. Такой способ передачи обеспечивает надежную передачу данных с линейной зависимостью между силой тока и измеряемым параметром, таким, как давление. Диапазон сигнала 4 – 20 мА стандартизирован: нижняя граница в 4 мА позволяет детектировать обрывы в цепи, а верхняя граница в 20 мА соответствует максимальному измеряемому значению [6]. Для их подключения к ПР200 необходимо выполнить настройку аналоговых входов как показано на рис.4.6.

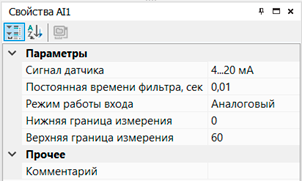


Рис.4.6. Настройки аналогового входа.

При настройке аналогового входа доступны следующие параметры:

1. Сигнал датчика – здесь можно выбрать то, каким образом датчик передает данные, используя напряжение, сопротивление или ток. В случае ПД100 используется токовая петля 4 - 20 мА.
2. Постоянная времени фильтра – данная настройка позволяет использовать аналоговый встроенный фильтр, тем самым убирая помехи на входе устройства. Фильтрация происходит по следующей формуле:

где

П – значение в регистре «Значение аналогового выхода»;

– значение, измеренное на входе;

– значение, измеренное на входе в предыдущий такт измерений;

T = 1 / (K / 10 + 1) – коэффициент сглаживания;

K – постоянная времени фильтра (сек).

Фактически это экспоненциальный фильтр. Оставим эту настройку стандартной.

1. Режим работы входа – эта настройка позволяет переключить аналоговый вход в дискретный, при необходимости, в нашем случае вход должен работать в аналоговом режиме.
2. Нижняя и верхняя границы измерений – настройки, которые позволяют автоматически преобразовать токовый диапазон 4 - 20 мА в диапазон нижняя - верхняя граница. Благодаря этим настройкам нет необходимости вручную обрабатывать данные тока, а сразу работать с показателями давления.

Поскольку в системе 2 датчика давления, эти настройки необходимо повторить для обоих аналоговых входов. Сохраним данные с датчиков в переменные, как показано на рис.4.7.

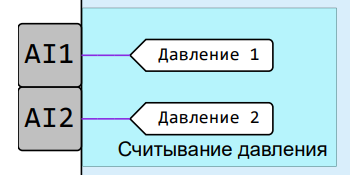


Рис.4.7. Считывание давления с аналоговых входов.

Аналогичные настройки зададим и для входа в ЭКП24, однако заменим токовую петлю на 0 – 10 В. Сохраним его значения в переменную.

Аналогично поступим с аналоговым выходом клапана, он не требует дополнительных настроек.

Далее рассмотрим протокол RS-485, он представляет собой стандарт физического уровня, который широко применяется в системах промышленной автоматизации из-за своей высокой надежности, устойчивости к помехам и способности обеспечивать стабильное соединение на значительных расстояниях [8]. В СИГ данный стандарт используется для соединения устройств СП310, ПЧВ и удаленного рабочего места оператора к ПР200.

Протоколом канального уровня для передачи данных по RS-485 будет Modbus. Это стандартная связка, которая часто используется на производствах и также реализована производителем в ПР200, СП310 и ПЧВ, однако при создании программы для удаленного рабочего места оператора придется реализовывать его обработку самостоятельно.

Modbus реализует архитектуру взаимодействия по модели «ведущий-ведомый». Взаимодействия в Modbus происходят единственным образом: ведущее устройство инициирует обмен данными, отправляя запросы одному или всем ведомым устройствам, которые предоставляют ответы. В контексте работы с СИГ, будет 2 Modbus сети – первая исключительно для работы с дисплеем СП310, где ведущим устройством будет ПР200, а вторая для работы с ПЧВ и удаленным рабочим местом оператора, где также ПР200 будет выступать ведущим устройством. Такое разделение вызвано большим количеством регистров, как у СП310, так и у удаленного рабочего места оператора, из-за этого в одной сети скорость их взаимодействия сильно падает. ПЧВ же можно было добавить в любую из сетей, однако с точки зрения прокладки проводов в щите было принято решение, что удобнее его добавить в сеть с удаленным рабочим местом оператора.

Выполним настройку Modbus в Owen Logic. Начнем с первого входа, который используется для связи с СП310, его настройки приведены на рис.4.8.

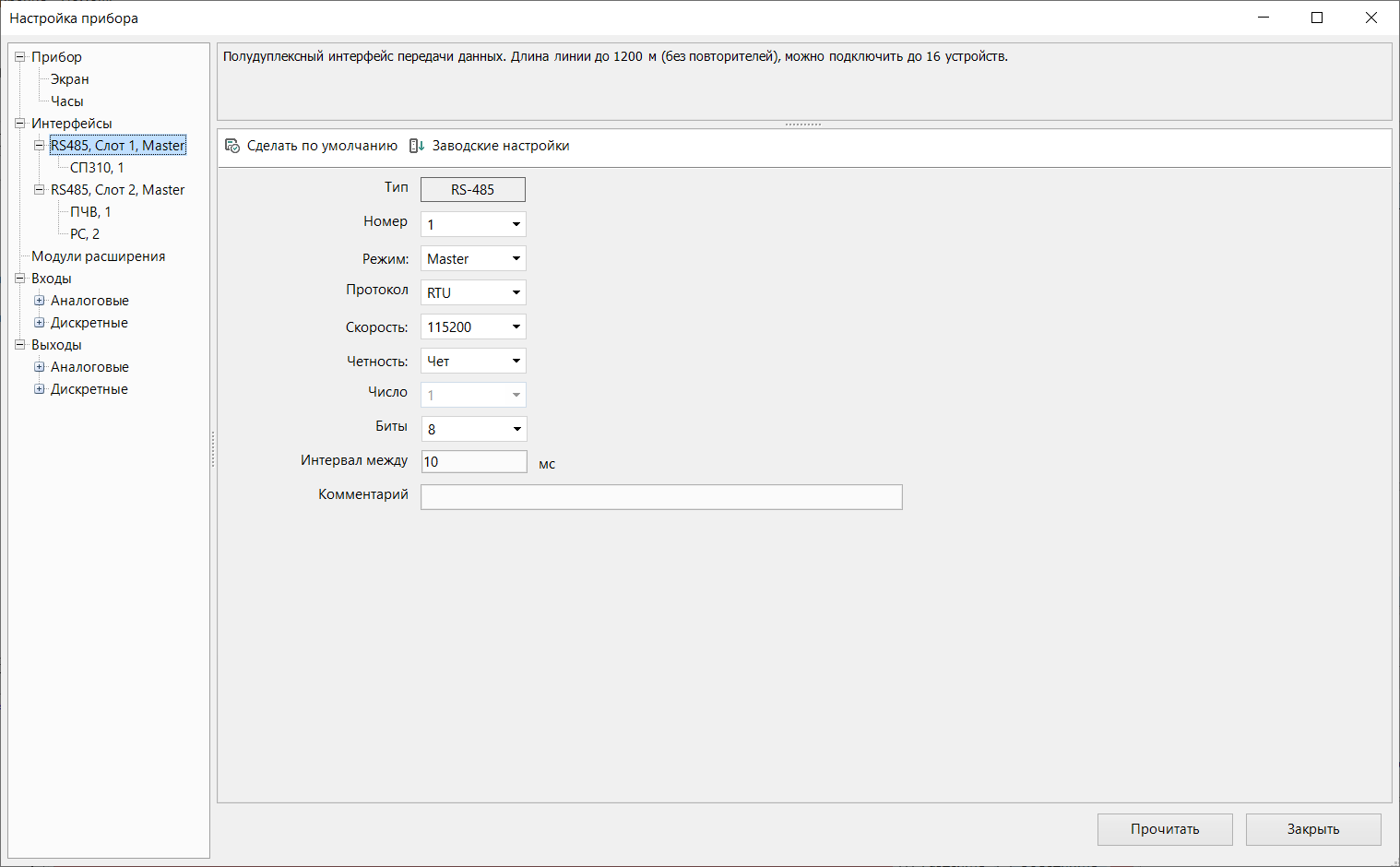


Рис.4.8. Окно настройки RS-485 для слота 1.

На рис.4.8. приведены следующие настройки:

1. Номер – это номер входа на ПР200, как и было сказано ранее, рассматриваем вариант для 1 слота.
2. Режим – здесь выбирается в каком режиме будет работать устройство, «Master» или «Slave». В СИГ ПР200 всегда «Master».
3. Протокол – в нашем случае выбирается RTU, также есть возможность работы в ASCII режиме, однако в этом нет необходимости.
4. Скорость – количество передаваемых бит в секунду. В контексте данной задачи 115200 бит/с было выбрано как максимальное значение для протокола RS-485, доступную для ПР200 и СП310.
5. Четность – позволяет включить бит четности. Для работы с СП310 установим «Чет», хотя это и не обязательно.
6. Число – количество битов четности. Стандартно 1.

Далее добавим устройство СП310. На рис.4.9 показаны его настройки регистры, которые будут использоваться в работе.

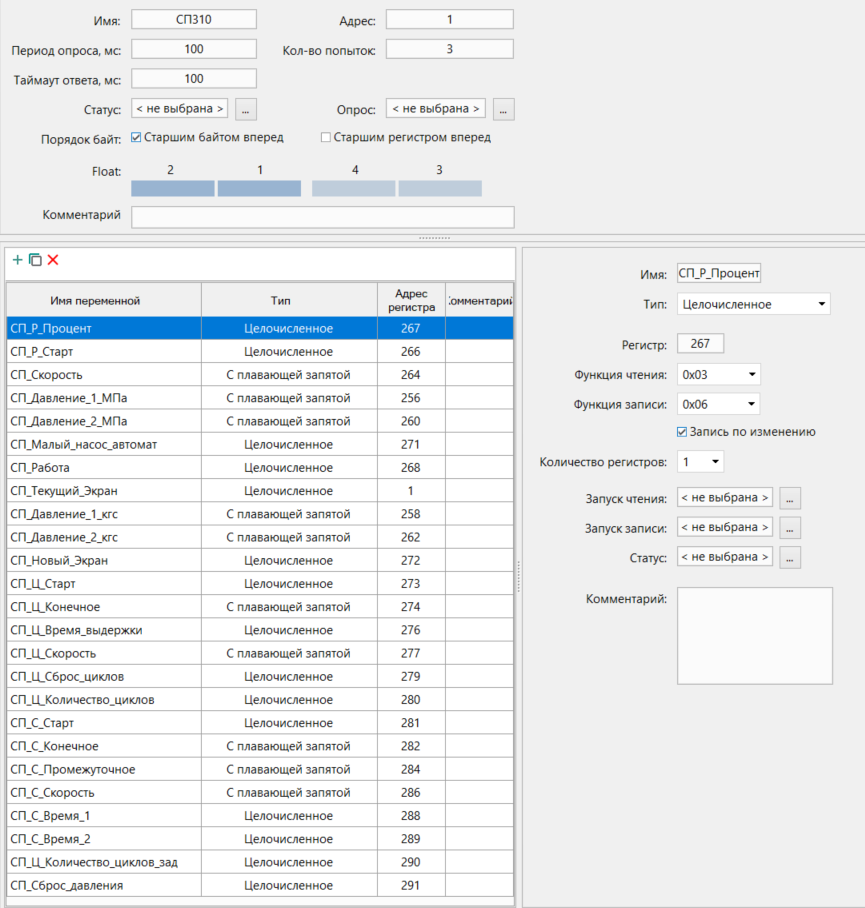


Рис.4.9. Окно настройки Modbus устройства для СП310.

Прежде всего рассмотрим настройки самого устройства, они расположены на рис.4.9 сверху.

1. Имя – то, какое имя будет отображаться в окне настроек устройства.
2. Адрес – Modbus адресный протокол, необходимо задать тот же адрес, который был задан в устройстве СП310. Поскольку СП310 будет настраиваться позже, можно выбрать любой.
3. Период опроса – то, с каким промежутком будут посылаться повторные попытки отправить команду в СП310. Поскольку Modbus имеет в себе механизм обратной связи, ПР200 может проверять, насколько успешно дошла посланная посылка. В случае, если обратная связь не пришла или пришла с ошибкой, повторный пакет отправится через период опроса миллисекунд. Оставим стандартное значение.
4. Кол-во попыток – то, сколько раз ПР200 будет пытаться повторно отправить посылку. Также оставим без изменений.
5. Таймаут ответа – то, через сколько считать посылку утерянной.
6. Статус и опрос – сервисные переменные, в нашем проекте не используются.
7. Старшим регистром вперед и старшим байтом вперед – настройки для передачи float чисел.

Помимо верхнего меню настроек справа снизу также есть меню. Это настройки уже переменной (регистра), который мы добавляем до чтения/записи. В нем есть следующие поля:

1. Имя – название этой переменной, которую мы будем использовать в основном поле Owen Logic после настройки.
2. Тип – задает тип переменной, целочисленный или с плавающей запятой.
3. Регистр – то, какой регистр будет опрашиваться у устройства.
4. Функция чтения и записи – в Modbus разные функции отвечают за разные действия (запись/чтение) и обращение к разным регистрам.
5. Запись по изменению – данный чекбокс показывает, будет ли посылаться команда на изменение сразу после изменения переменной в ПР200 или ожидать сигнала. В нашем случае запись будет производиться сразу.
6. Запуск чтения и запуск записи – позволяет выбрать переменные, которые при установке значения 1 будут читать или записывать значение этой переменной. В данном проекте это не используется.
7. Статус – позволяет добавить переменную, которая будет показывать статус. Может помочь при отладке в случае, если переменная не функционирует в соответствии с ожиданиями.

Для второго входа, настройки приведены на рис.4.10.

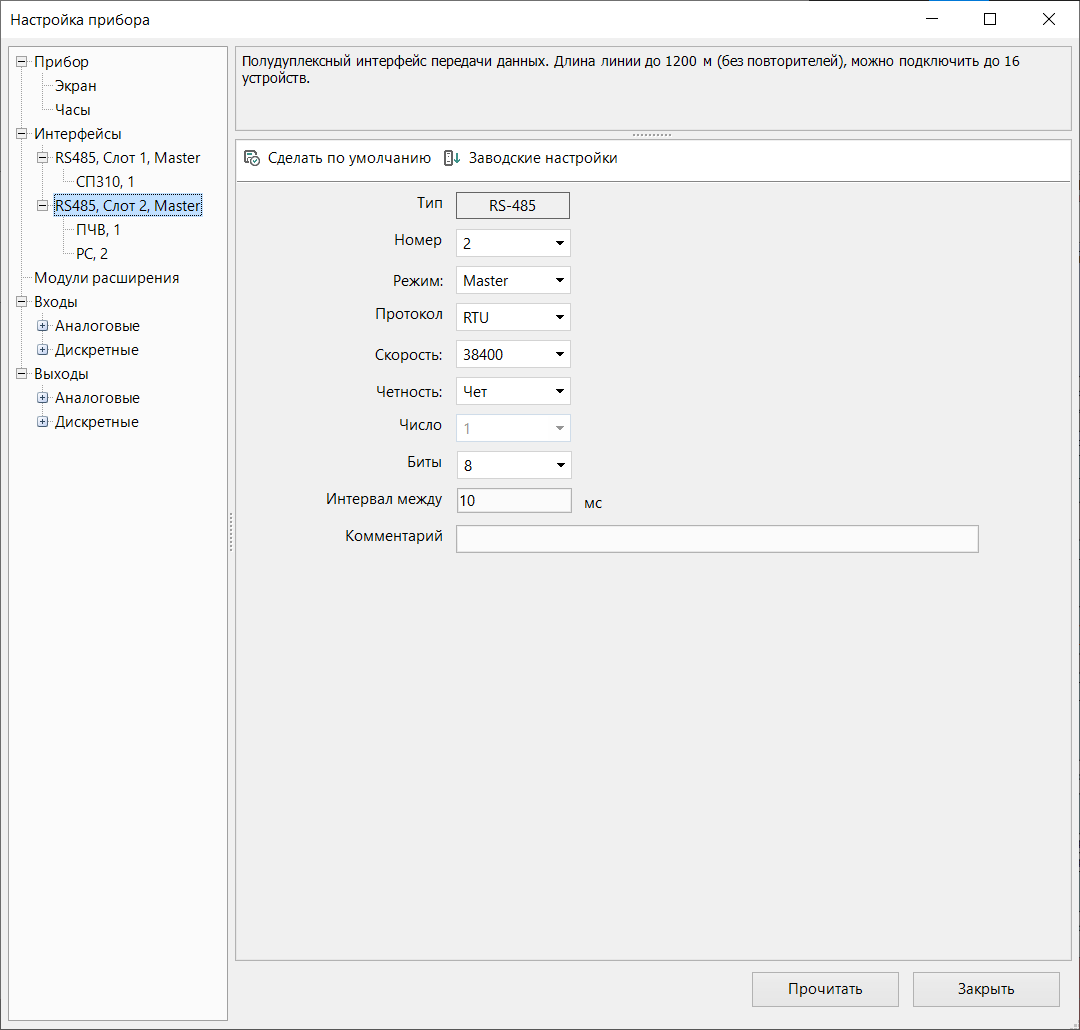


Рис.4.10. Окно настройки RS-485 для слота 2.

К нему подключено 2 устройства, а именно ПЧВ (настройки на рис.4.11) и удаленное рабочее место оператора (настройки на рис.4.12).

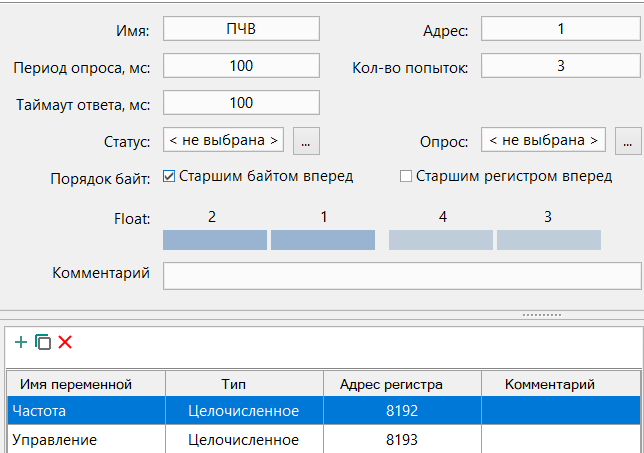


Рис.4.11. Окно настройки Modbus устройства для ПЧВ.

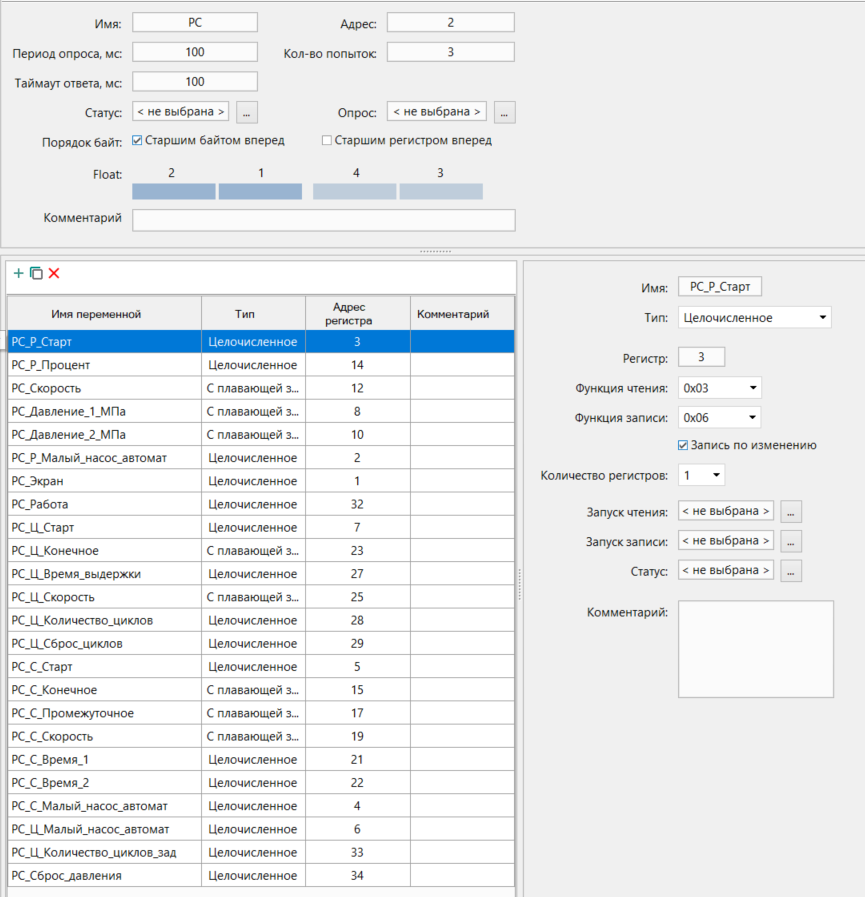


Рис.4.12. Окно настройки Modbus устройства для удаленного рабочего места оператора.

Таким образом к ПР200 были подключены все периферийные устройства.

## Разработка программы для СП310

Прежде всего начнем с программы для экрана оператора СП310. Как и было сказано ранее, разработка ведется в программе «Конфигуратор СП300», разработанной специально для панелей оператора компании ОВЕН. Его внешний вид приведен на рис.4.13.

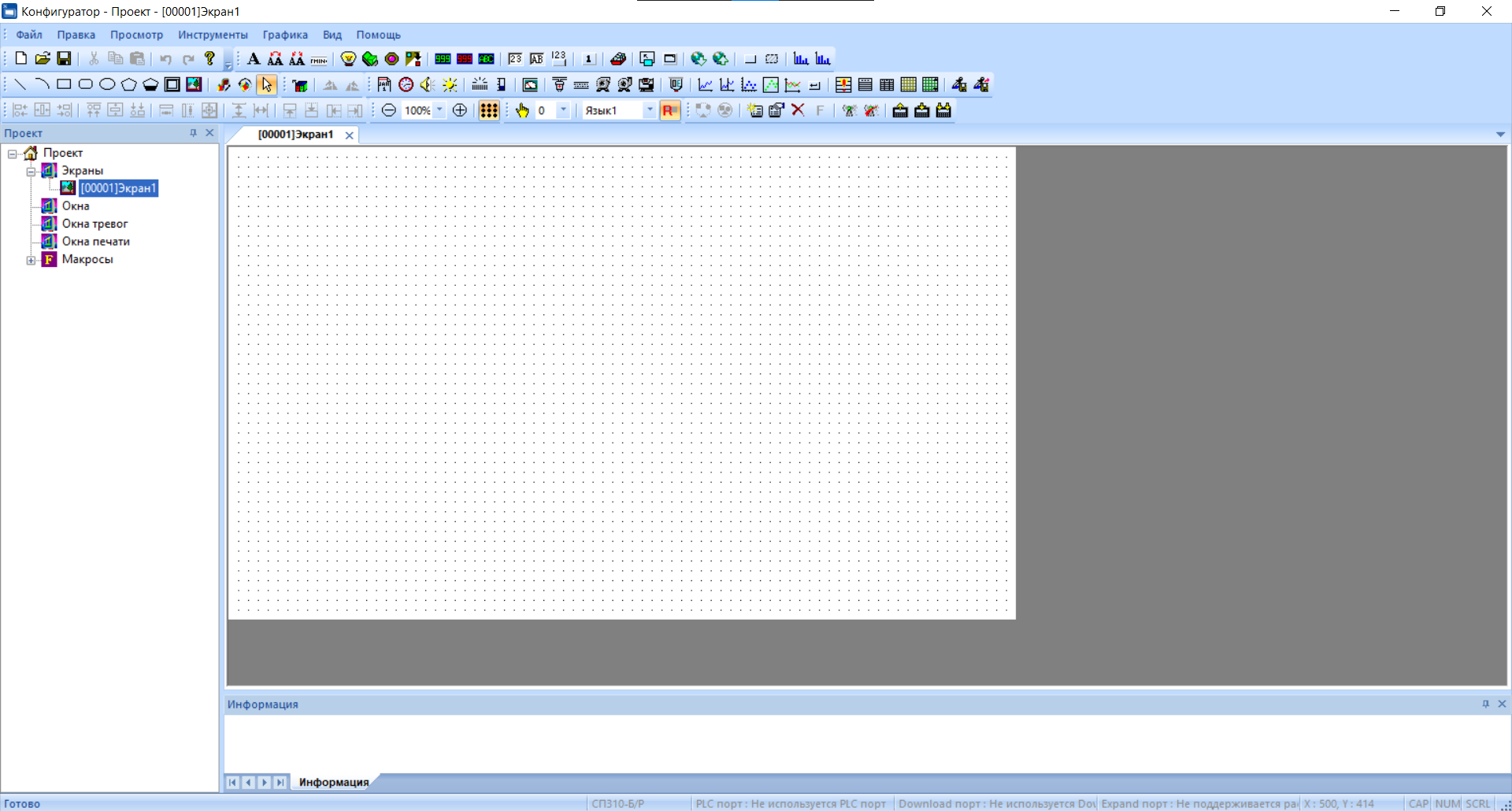


Рис.4.13. Окно программы «Конфигуратор СП300».

Как видно по рис.4.13, это графическая среда разработки. Выполним конфигурацию порта для связи с ПР200, эти настройки будут идентичны тем, которые задавались ранее для соответствующего порта в ПР200. Данные настройки приведены на рис.4.14.

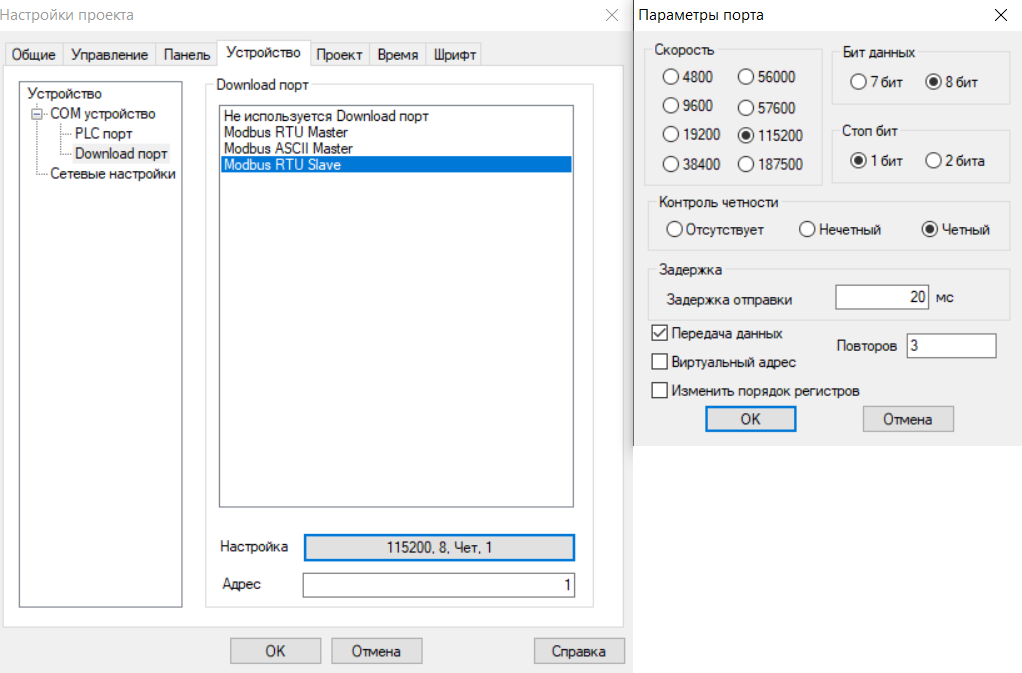


Рис.4.14. Настройки порта связи в СП310.

На рис.4.15 представлена диаграмма переходов между экранами, которые необходимо реализовать.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рис.4.15. Диаграмма переходов между экранами.

Выполним создание экранов, основываясь на рис.4.15, а также описании алгоритмов из главы 3. Итоговый результат приведен в Приложение .

Созданные экраны имеют свои уникальные ID (в квадратных скобках перед названием экранов на рис.4.16), которые используются для идентификации режима, в котором находится пользователь.

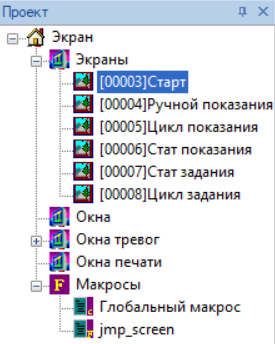


Рис.4.16. Экраны и макросы проекта.

В панели оператора есть стандартный регистр, в котором находится текущий номер экрана (регистр 1), однако он доступен только для чтения. В следствии этого было необходимо создать макрос, который бы позволял синхронизировать экраны на удаленном рабочем месте оператора и в панели оператора. Для этого был создан макрос, jmp\_screen, его код приведен в листинге 4.1.

Листинг 4.1. Макрос jmp\_screen.

|  |
| --- |
| ScreenJump(PSW[272]); |

Этот макрос изменяет текущий экран на ID, указанный в регистре внутренней памяти под номером 272. Для его непрерывной работы была создана глобальная функциональная область (её видно справа снизу на каждом экране в Приложение ) её настройки приведены на рис.4.17.

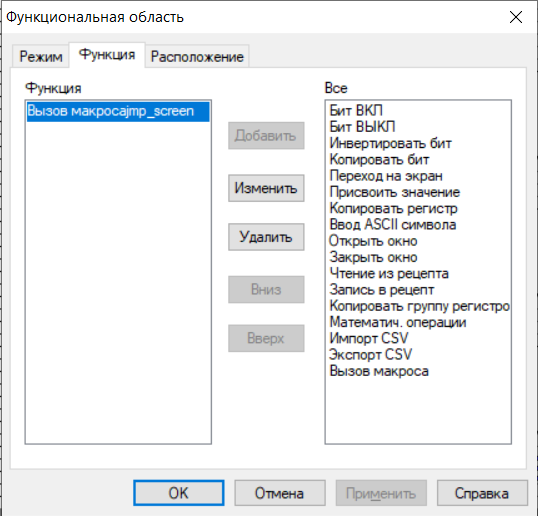
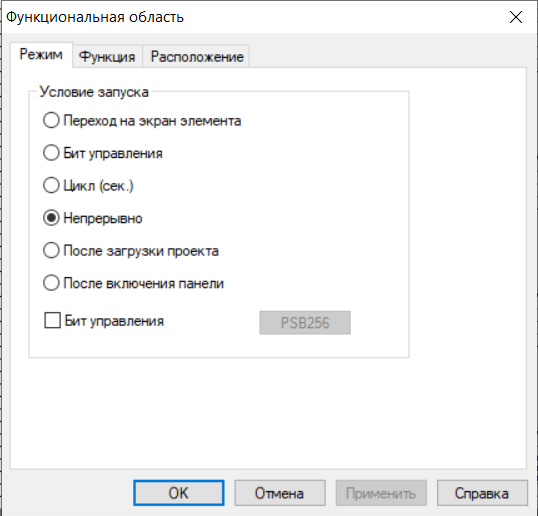


Рис.4.17. Настройки функциональной области.

В панели оператора существует возможность подключения USB флеш-накопителя с целью логирования какой-то информации. Воспользуемся этой возможностью и будем сохранять текущее давление каждую секунду, для этого выполнена настройка области «Архивирование USB» (её видно справа снизу на каждом экране в Приложение ), как показано на рис.4.18.

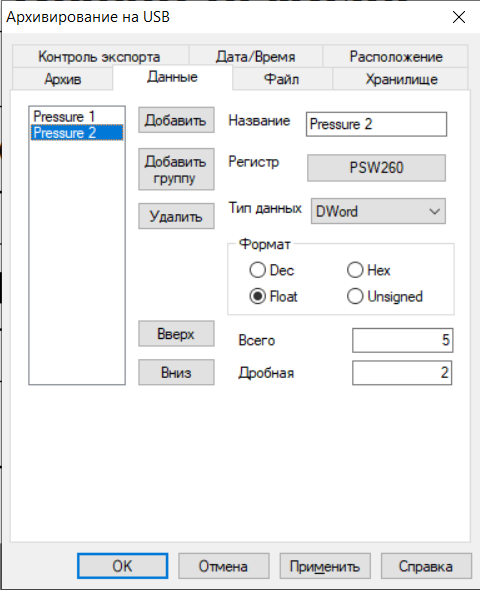
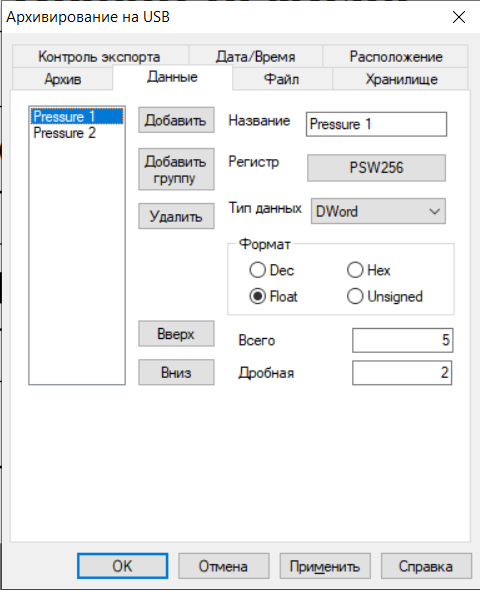
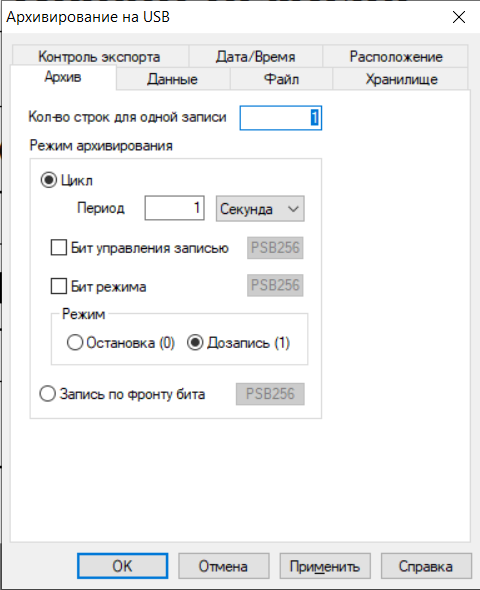


Рис.4.18. Настройки области «Архивирование USB».

Регистры давления выбраны в соответствии с регистрами, указанными на рис.4.9, в дальнейшем все остальные регистры панели оператора также будут указаны в соответствии с данным рисунком.

Функциональные области, приведенные выше, не будут отображаться на экранах, а будут автоматически скрыты т.к. не несут пользователю никакой информации. Далее рассмотрим каждое окно панели оператора по отдельности.

Перове окно, которое появляется при загрузке панели – окно выбора режима (приведено на рис.П3.1). На нем расположено 3 кнопки с разным режимами, которые ведут в другие окна, в соответствии с диаграммой на рис.4.15. Данные кнопки являются стандартными переходами, без каких-либо дополнительных условий, поэтому данное окно не представляет интереса.

Следующим рассматриваемым окном будет окно ручного режима (приведено на рис.П3.2). Здесь рассмотрим сразу же несколько элементов.

Начнем с кнопки «Выбор режима», она возвращает к окну выбора, однако она пропадает, когда включен ПЧВ. Это сделано для того, чтоб оператор не мог уйти из режима прямо во время работы установки. Это сделано, используя регистр 268, который будет задаваться ПР200 и функцией видимости, которая приведена на рис.4.19.

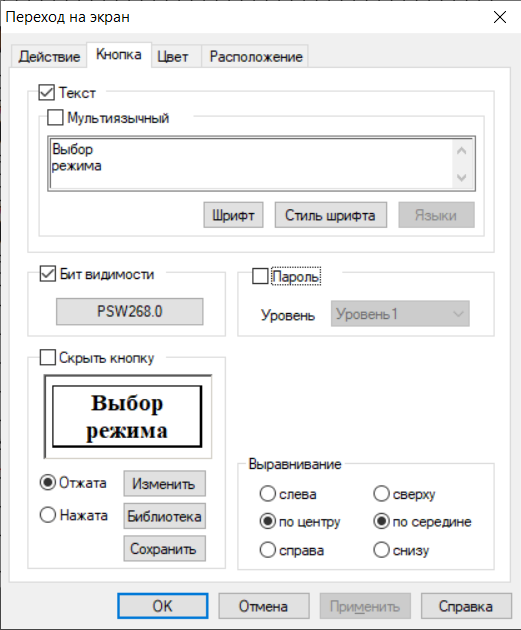
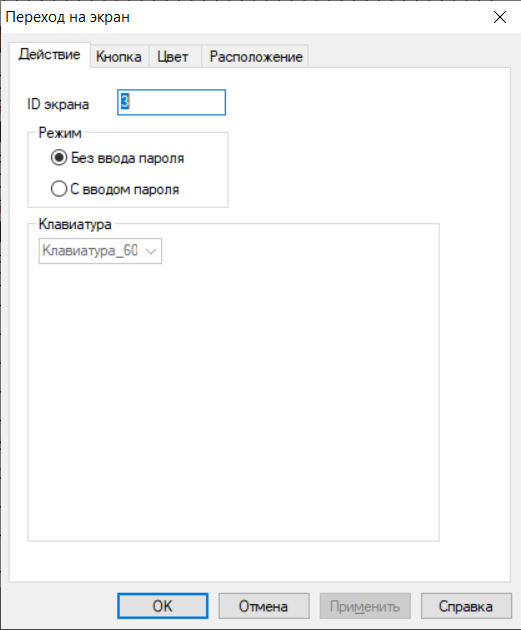


Рис. .. Настройки кнопки «Выбор режима».

Поля с выводом значения давления являются стандартными и показывают соответствующие им регистры (256 и 260 для МПа и 258 и 262 для кгс/см2), также как и поле скорости (264 регистр).

Частота является вводимым параметром, что отражает соответствующая черная рамка. Используется стандартное поле ввода с регистром 267.

Кнопки запуска насосов и сброса давления – двухпозиционные, настроенные на изменение и наблюдение (для изменения своего состояния при изменении регистра через Modbus) соответствующих регистров (271 для малого насоса, 266 для большого насоса, 291 для клапана сброса давления).

Последний оставшийся элемент – график. Его настройки приведены на рис.4.20.

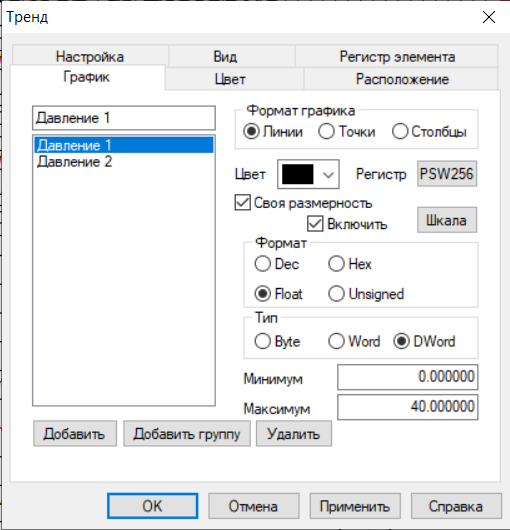
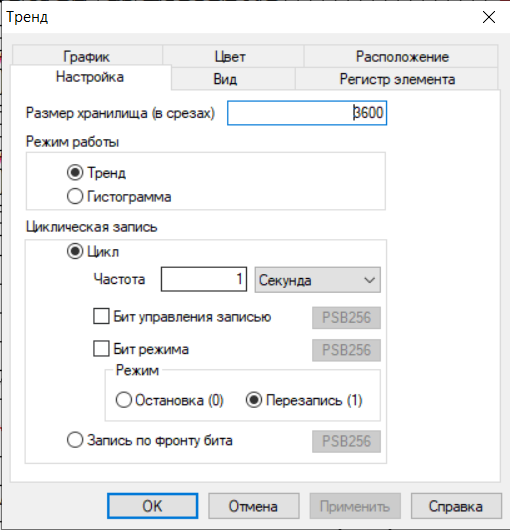


Рис.4.20. Настройки графика.

Он хранит информацию о последнем часе испытаний и выводит последние 10 минут, стрелки влево и вправо помогают передвигаться по нему, а кнопка сброса возвращает график в исходное положение.

Элементы на экране управления статическим режимом (приведен на рис.П3.6) идентичны, однако отсутствует поле задания текущей частоты работы двигателя т.к. он выбирается автоматически.

На экране управления циклическим режимом (приведен на рис.П3.4) дополнительно присутствует поле текущего числа выполненных циклов, он нужен для удобства оператора.

Экран настроек статического режима (приведен на рис.П3.5) представляет собой 5 полей, в которые необходимо вписать данные в соответствии с пунктом 3.4. Валидация введенных данных должна происходить на стороне ПР200, СП310 только дает возможность ввода. Важно, чтоб эти данные дублировались между СП310 и удаленным рабочим местом оператора, однако это задача также будет решаться, используя ПР200.

Экран настроек циклического режима (приведен на рис.П3.3) также представляет собой 4 поля для ввода в соответствии с пунктом 3.3. Нижняя граница, до которой должно опускаться давление будет задаваться с экрана ПР200 т.к. обычно это значение не изменяется на испытаниях. Особенность этого экрана – кнопка для сброса текущего числа циклов, она необходима для удобства оператора. Её уникальность в том, что обычно устройства управления являются ведущими, а управляемые устройства – ведомыми, но здесь все наоборот, поэтому возможности послать информацию ПР200 о том, что кнопка нажата - возможности нет, в следствии этого был реализован следующий алгоритм: кнопка устанавливает соответствующий ей регистр (279) в единицу, после чего ПР200 считывает его и самостоятельно его сбрасывает.

Таким образом были рассмотрены все окна и функциональные элементы в СП310.

## Разработка программы для удаленного рабочего места оператора

Далее необходимо разработать программу для дублирования панели оператора для удаленного рабочего места оператора. Она существует для того, чтоб вынести место работы оператора на время работы установки в другое помещение т.к. вблизи установки работать может быть небезопасно.

Программа для удаленного рабочего места оператора должна полностью дублировать функционал СП310, рассмотренный ранее, а именно отслеживание текущих параметров СИГ, возможность редактирования настроек режима, включение и выключение режима. Переключение окон должно полностью дублироваться между СИГ и СП310.

Разрабатываемая программа написана на языке python версии более 3.12, использует библиотеку tkinter для реализации графического интерфейса. Она состоит из следующих модулей:

* main.py – файл верхнего уровня для запуска программы.
* App.py – главный файл программы, в нем создается основное окно программы и все переключаемые окна, настраиваются все функции обратного вызова, а также создается модуль для работы с Modbus.
* ModbusSlave.py – непосредственно модуль для работы с Modbus, занимается поиском порта, на котором находится протокол, обрабатывает все взаимодействия с ним и хранит карту регистров, которая была показана на рис.4.12.
* frames – набор модулей, каждый из которых наследуется от BaseFrame, в котором реализованы основные функции экранов. В этом наборе модулей содержатся все созданные экраны.

Пройдемся по очереди по каждому модулю. Код файла верхнего уровня приведен в листинг П3.1. Как и было сказано выше, данный файл создает само приложение и запускает его.

Далее рассмотрим листинг П3.2. В нем содержится класс App, который является непосредственно самим приложением, запускаемым в файле main.py. Список его методов приведен на рис.4.21.

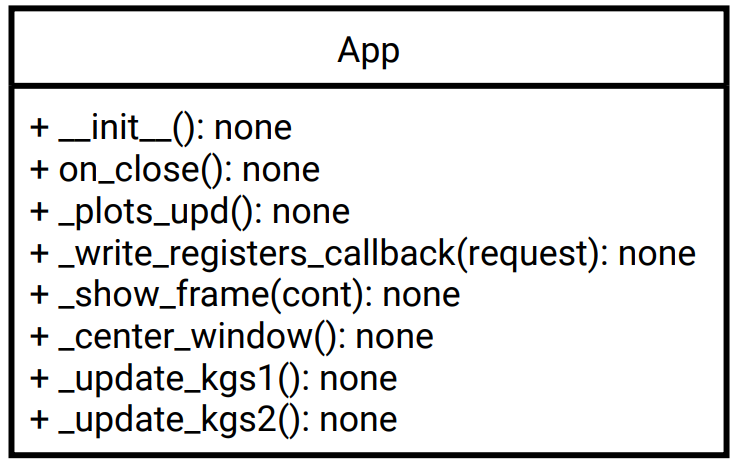


Рис.4.21. Описание класса App.

Функция \_\_init\_\_, вызываемая при создании экземпляра класса, задает шрифты и размеры для окна приложения, создает карту экранов и сами экраны, между которыми будет переключаться пользователь, настраивает Modbus, описывает функции обратного вызова, чтоб при записи новых значений в регистры, они отображались на экране.

Функция on\_close, которая вызывается при закрытии программы, уничтожает открытое окно, а также заканчивает опрос открытого com-порта.

Остальные функции вспомогательные и вызываются в описанных ранее:

* \_plots\_upd – функция, которая вызывается раз в секунду и обновляет графики на окнах.
* \_write\_registers\_callback – функция для обработки записи значений в кнопки или изменения экрана.
* \_show\_frame – функция, служащая для изменения текущего фрейма на другой, переданный ей в качестве параметра.
* \_center\_window – как понятно из названия, центрирует окно с приложением.
* \_update\_kgs1, \_update\_kgs2 – две функции, которые преобразуют значение МПа в кгс/см2.

Перейдем к рассмотрению листинга П3.3 в нем приведено описание класса ModbusSlave, который отвечает непосредственно за обработку Modbus посылок, его структура приведена на рис.4.22.



Рис.4.22. Описание класса ModbusSlave.

Прежде чем перейти непосредственно к рассмотрению методов класса, рассмотрим Modbus. В полной его реализации в Modbus существует несколько областей памяти, которые (по необходимости разработчика) могут накладываться друг на друга. Эти области приведены в Таблица 4.1. Однако такое разделение памяти в данном проекте излишне, как и рекомендуемая нумерация регистров, в данном проекте будет только 1 тип памяти, а именно последний, нумерация будет начинаться с 1.

Таблица 4.

Области памяти в Modbus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  регистра | Адрес регистра HEX | Тип | Название | Тип |
| 1-9999 | 0000 до 270E | R/W | Discrete Output Coils | DO |
| 10001-19999 | 0000 до 270E | R | Discrete Input Contacts | DI |
| 30001-39999 | 0000 до 270E | R | Analog Input Registers | AI |
| 40001-49999 | 0000 до 270E | R/W | Analog Output Holding Registers | AO |

Также необходимо рассмотреть структуру посылки Modbus, которая имеет формат посылки, приведенный на рис.4.23.

|  |  |  |  |
| --- | --- | --- | --- |
| Slave ID | Код функции | Специальные данные | CRC |

Рис.4.23. Общая структура посылки Modbus.

Первый байт – адрес устройства, к которому идет обращение. Если адрес не совпадает, устройство должно игнорировать посылку.

Второй байт – код функции, то, какую команду должно исполнять устройство, в данном случае валидными считаются:

* 0x03 – чтение регистров.
* 0x06 – запись одного регистра.
* 0x10 – запись нескольких регистров.

Последние 2 байта – CRC16, с полиномом 0xA001, и стартовым значением 0xFFFF.

Остальная же посылка уникальная для каждого конкретного кода функции и того, что мы формируем, запрос или ответ. Для команд 0x03, 0x06, 0x10 приведены соответствующие форматы в Приложение .

После рассмотрения команд Modbus перейдем к непосредственно коду, первым идет метод \_\_init\_\_, при создании экземпляра класса создаются служебные поля, а также массив внутренних регистров, как было ранее сказано, это только один массив Holding Registers.

Далее идет метод start, как понятно из названия, она запускает процесс обработки, а именно автоматически находит подходящий com-порт, после чего переходит к процессу его «прослушивания», обрабатывая приходящие команды. Процесс обработки запускается в отдельном потоке для того, чтоб не тормозить пользовательский интерфейс.

Следующий метод – stop. Он выключает опрос и закрывает порт для опроса.

Последний основной метод – set\_callback, он нужен для того, чтоб добавить обработчики из основного класса, в этот, т.е. чтоб на экране происходили изменения после прихода команд Modbus.

Остальные функции являются вспомогательными, они занимаются обработкой самих команд, вычислением CRC, расчетом длины посылок и формированием ответов на запросы.

Последним ключевым компонентов программы являются сами окна, их код приведен с листинга П3.3 по листинг П3.9, как было сказано ранее, они все наследуются от базового окна, схема приведена на рис.4.24.

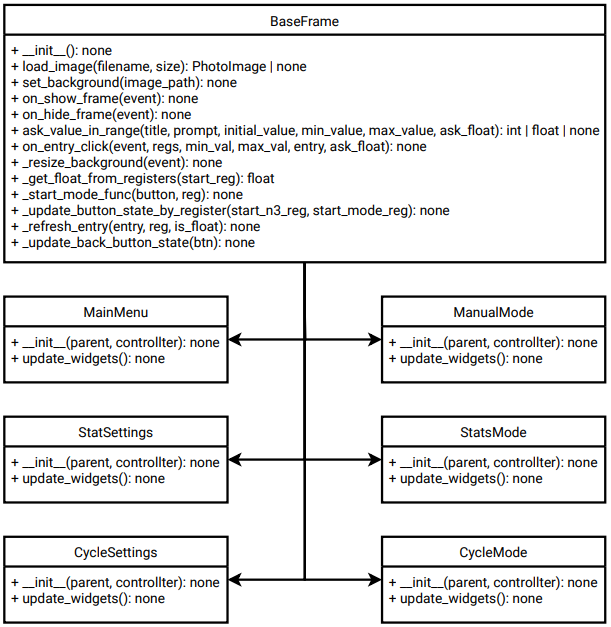


Рис. .. Описание классов окон.

Все наследники имеют только 2 функции. В \_\_init\_\_ происходит отрисовка самого окна, подключение всех необходимых зависимостей для обработки взаимодействий с регистрами Modbus. Во второй же функции update\_widgets происходит обновление состояния в соответствии с регистрами, эту функцию вызывает App при обновлении информации.

Основной же класс BaseFrame предоставляет наследникам функции для создания фона, основываясь на картинках (это сделано для того, чтоб интерфейс был похож на аналогичный в СП310), обрабатывает открытие и закрытие окон, а также предоставляет функции-обработчики для некоторых общих кнопок (например кнопок назад в каждом из режимов, которые блокируются при запуске режима).

Полная версия проекта доступна в [9].

## Разработка программы для ПР200

Перейдем к разработке программы для ПР200. На данный момент выполнены подключения всех периферийных устройств.

Первым делом настроим передачу давления в СП310 и в удаленное рабочее место оператора, для этого подключим необходимые данные с входов в соответствующие регистры (см рис.4.25).

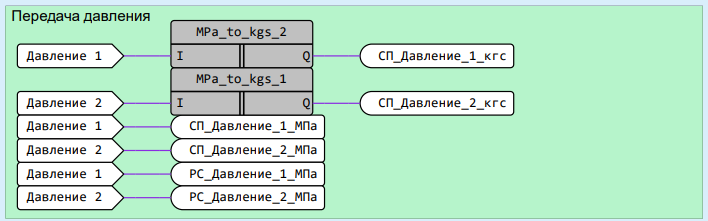


Рис.4.25. Передача давления из ПР200.

Как говорилось ранее, в удаленном рабочем месте оператора давление преобразуется автоматически, однако в СП310 данной возможности нет, поэтому преобразование в кгс/см2 необходимо проводить в ПР200, для этого была создана функция MPa\_to\_kgs\_, её код приведен в листинге 4.2.

Листинг 4.2. Код функции MPa\_to\_kgs\_.

|  |
| --- |
| function MPa\_to\_kgs\_: real;   var\_input  I : real;  end\_var    MPa\_to\_kgs\_ := I \* 10.197162;   end\_function |

Далее рассмотрим управление ПЧВ на рис.4.26.

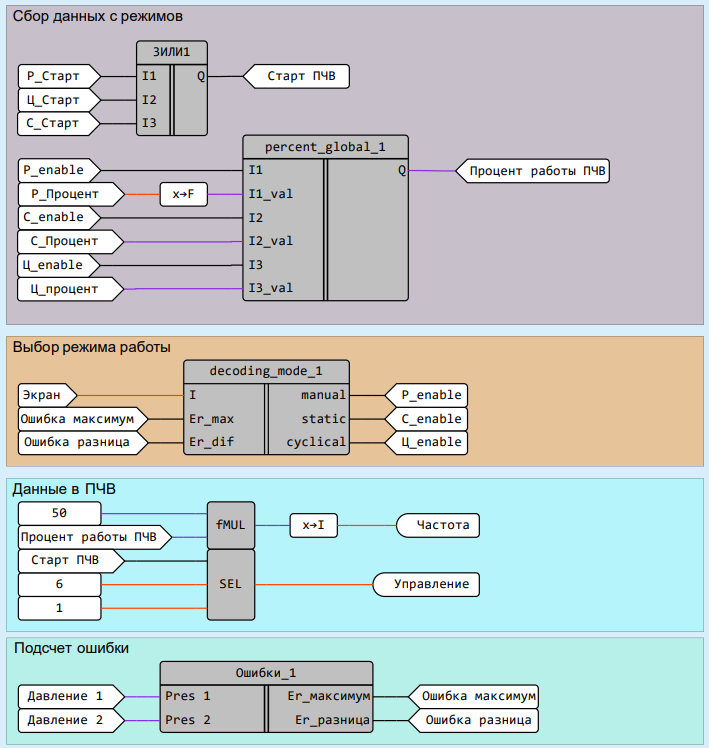


Рис.4.26. Управление ПЧВ.

ПЧВ стартует если приходит старт из любого из режимов, процент выбирается на основании того, какой режим сейчас на экране СП310 и удаленном рабочем месте оператора. ПЧВ не запускается, если произошла ошибка по превышению давления в установке, либо между датчиками давления большая разница, причиной этого может служить закрытый клапан (см рис.3.1), либо ошибка самого датчика. По ошибке разницы можно запустить ручной режим, но любой другой режим работать не будет.

Очень важным блоком является блок синхронизации экранов между СП310 и удаленным рабочем местом оператора, для этого используется блок на рис.4.27.

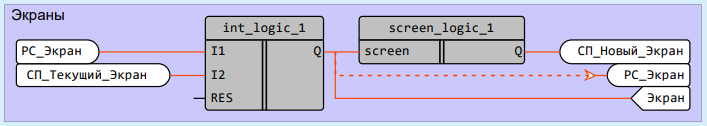


Рис.4.27. Синхронизация экранов на СП310 и  
удаленном рабочем месте оператора.

Функциональный блок int\_logic\_ будет использоваться в проекте достаточно часто. Он служит дополнительной задержкой при изменении величин через Modbus, фактически он ждет фронт изменения величины и останавливает изменение выходной на 3 такта, чтоб этот фронт дошел также и до остальных величин, его код приведен в листинге П5.1.

Далее только в СП310 идет дополнительный блок обработки, он выставляет новый экран на 100 ms, а затем выдает 0. Сделано это потому, что для СП310 первичен экран, находящийся в переменной, из-за этого переключение экранов не происходит, однако при записи нуля, СП310 игнорирует перемещение на этот экран, поэтому позволяет свободно переключать окна.

Последним общим блоком для всех экранов остается подсчет скорости (приведен на рис.4.28), он выполняется исключительно на изменении величины «давление 2» т.к. она наиболее близко расположена к баку и предоставляет самые актуальные данные.

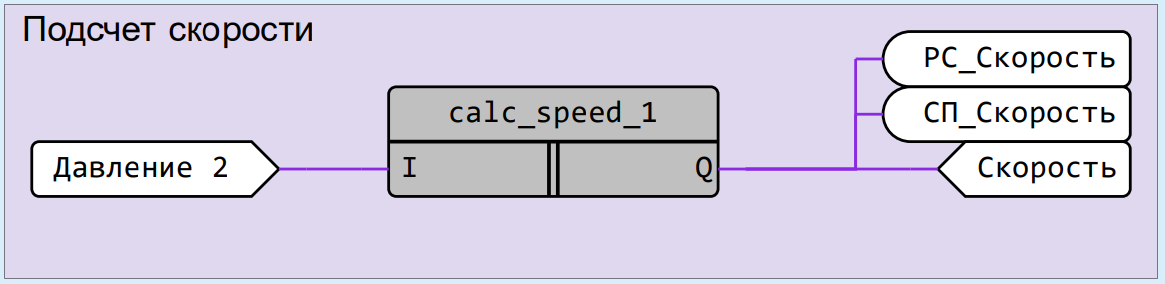


Рис. .. Подсчет скорости.

В качестве алгоритма для подсчета скорости используется фильтр Савицкого-Голея, это линейный цифровой фильтр с конечной импульсной характеристикой, основывающийся на скользящем окне фиксированной длинны, где исходная выборка точек аппроксимируется многочленом заданной степени. Результатом фильтра является значение или оценка его производной в центре окна [4]. Такой подход позволяет эффективно подавлять шум, а также быстро вычислять производную, что нам и требуется.

В общем виде формулы для подсчета фильтра выглядят следующим образом [4]:

где – значение или оценка производной;

– значения измеренной величины;

– коэффициенты фильтра, которые вычисляются по следующей формуле:

где – матрица Вандермонда, окна, ; m – порядок полинома, s – порядок производной;

– единичный столбец, длинной , с 1 на позиции s;

– вектор свёрточных коэффициентов для производной порядка s.

В данном проекте m = 1, т.е. будет линейная аппроксимация т.к. она максимально устойчива к шумам, а s = 1 т.к. необходима первая производная. Тогда формула вычисления коэффициентов упростится до следующего вида:

где h – шаг дискретизации (сек);

– нормирующий коэффициент,

– вес для оценки

Будем считать скорость за 2 секунды работы установки, шаг дискретизации возьмем 200 мс, тогда фильтр будет длиной 10 точек. Длина фильтра должна быть нечетной, возьмем 9 точек (1.8 секунд), тогда . Тогда коэффициенты фильтра будут следующие:

Вышеописанный алгоритм на языке ST приведен в листинге П5.5.

Теперь перейдем к рассмотрению алгоритма работы малого насоса. Т.к. он может работать независимо от режима, было решено вынести его отдельно от всех режимов. Его структура приведена на рис.4.29.

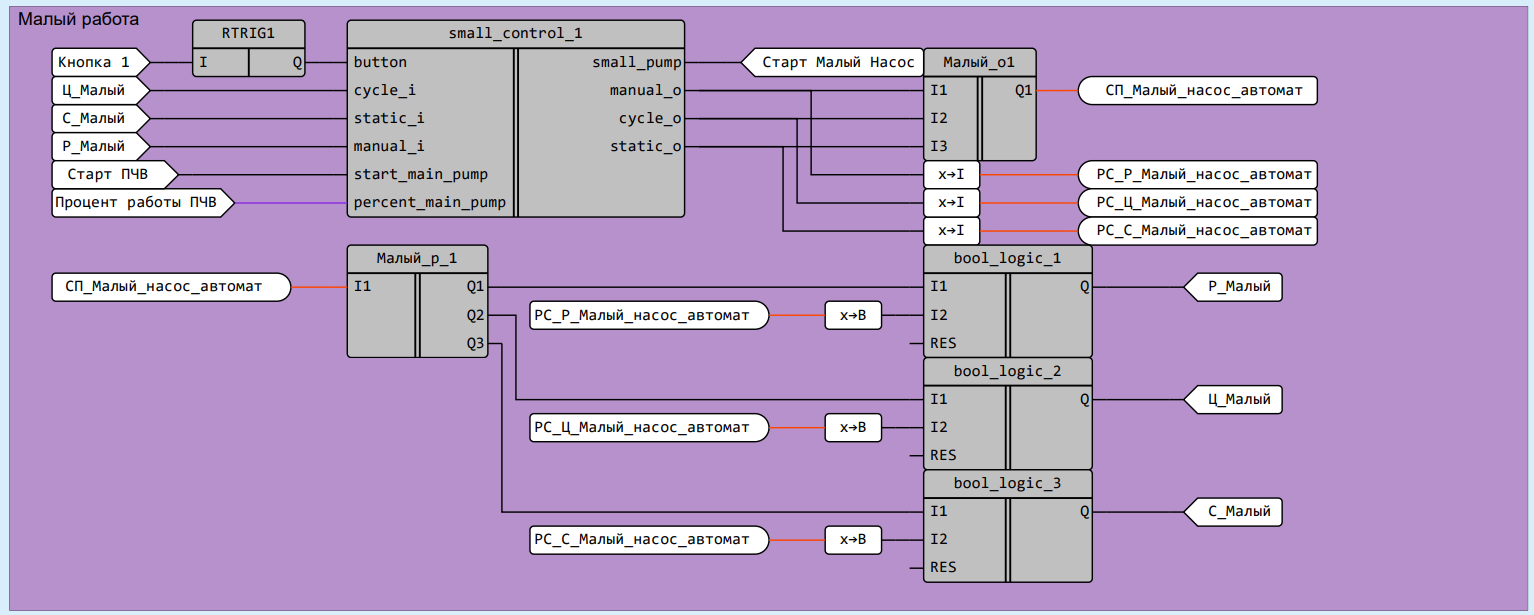


Рис.4.29. Код малого насоса в блоковых диаграммах.

Поскольку в СП310 за малый насос отвечает один регистр, но с разными битами, а в удаленном рабочем месте оператора это три полноценных разных регистра код выглядит не симметрично, как в остальных местах. Нижние 4 блока отвечают за определение того, какой режим включил насос, после чего эта информация отправляется в основной блок обработки работы с малым насосом, также туда отправляется 1 кнопка на ПР200, которая также отвечает за малый насос.

Код данного блока приведен в листинге П5.2. Алгоритм работы достаточно прост, если основной насос не включен, тогда происходит стабильное наполнение бака водой, однако если уже включен рабочий режим, то малый насос работает с интервалами. Необходимость в данном алгоритме заключается в том, что малый насос подает воду куда быстрее, чем большой, в связи с этим в части, в которой не должно быть давление оно набирается, дополнительно тратится ресурс малого насоса. В связи с этим было принято решения не держать малый насос включенным постоянно, а включать и выключать его интервалами, но только во время работы в режиме.

Далее перейдем к рассмотрению реализации первого режима работы – ручного. Он состоит из 4 основных блоков, первым является блок активации кнопок (см рис.4.30) её можно произвести только, если пользователь находится на экране ручного режима. Дополнительно присутствует блок защиты, который запрещает одновременное нажатие 3 и 4 кнопок, т.к. обе эти кнопки отвечают за включение ПЧВ на определенный процент работы, что описано в разделе 3.2.

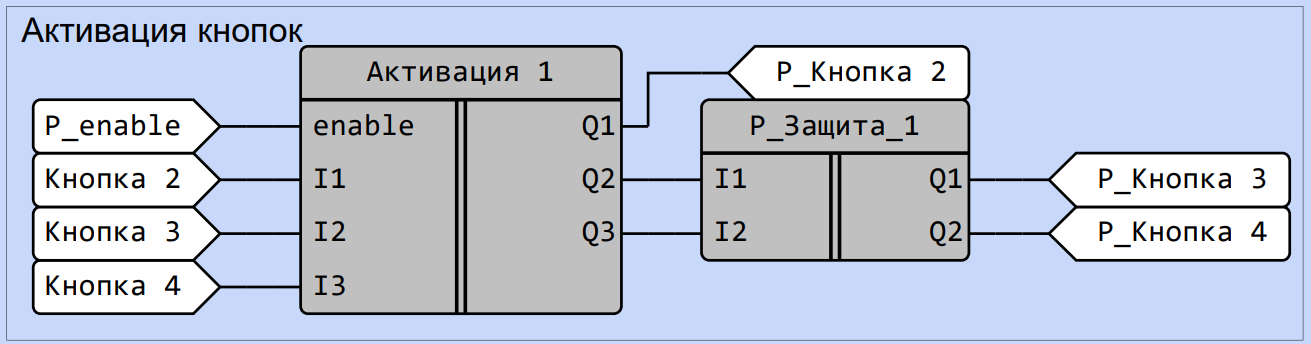


Рис.4.30. Активация кнопок ручного режима.

Следующим блоком является блок обработки кнопок (см. рис.4.31), он реализует алгоритм обработки кнопок, описанный в разделе 3.2.



Рис.4.31. Блок обработки кнопок.

Функциональный блок ON\_OFF\_ST\_ занимается непосредственно обработкой длительности нажатия и фиксации установочной частоты. Его код приведен в листинге П5.3.

Вспомогательные переменные Р\_Задать процент и Р\_Установочная используются в следующем блоке (см рис.4.32), занимающимся обработкой синхронизации процентов полученных при нажатии кнопок, с экрана СП310 и с удаленным рабочим местом оператора.



Рис.4.32. Блок обработки итоговой частоты ручного режима.

Последним блоком в ручном режиме является блок запуска ПЧВ, он приведен на рис.4.33. Элемент button\_logic\_ аналогичен по логике и сути работы элементу int\_logic\_, но работает для bool входов и для трех элементов, его код приведен в листинге П5.4.

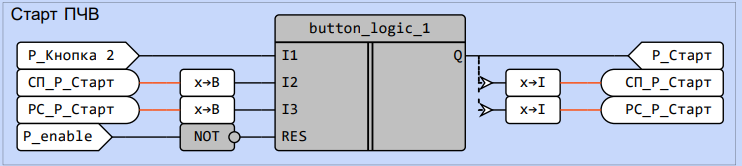


Рис. .. Блок запуска ПЧВ в ручном режиме.

Перейдем к рассмотрению автоматических режимов управления, первым рассмотрим циклический режим, он состоит из 6 блоков, 2 из них похожи на аналогичные из ручного режима: для активации кнопок и запуска ПЧВ (см.рис.4.34), отличие в том, что ПЧВ теперь должен выключаться по завершению работы режима, а также 3 и 4 кнопки не используются, поэтому нет необходимости их активировать и включать дополнительную защиту.

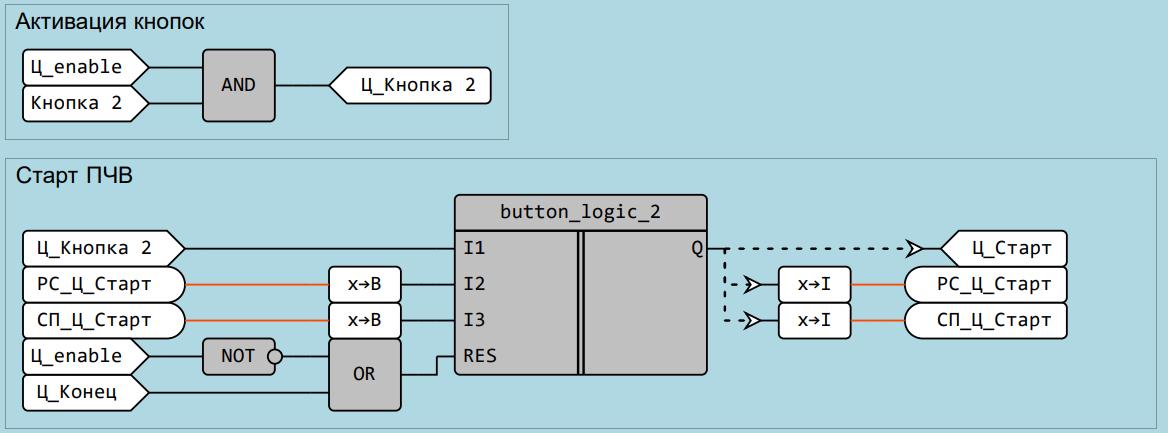


Рис.4.34. Блок запуска ПЧВ и активации кнопок в циклическом режиме.

Еще 2 блока тоже простые, они отвечают за передачу циклов и сброс состояний кнопок сброса, чтоб они могли работать повторно (см. рис.4.35).

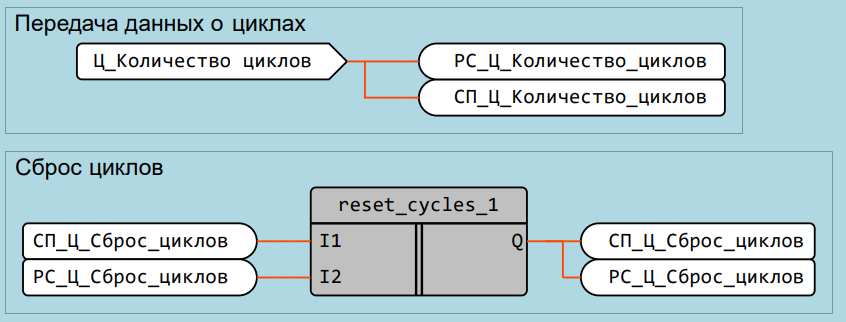


Рис.4.35. Блок передачи циклов и обработки их кнопок сброса.

Следующим блоком этого режима является обработка вводимых значений (см. рис.4.36), как понятно из названия он проверяет корректность вводимых значений, а также синхронизирует введенные параметры между устройствами.

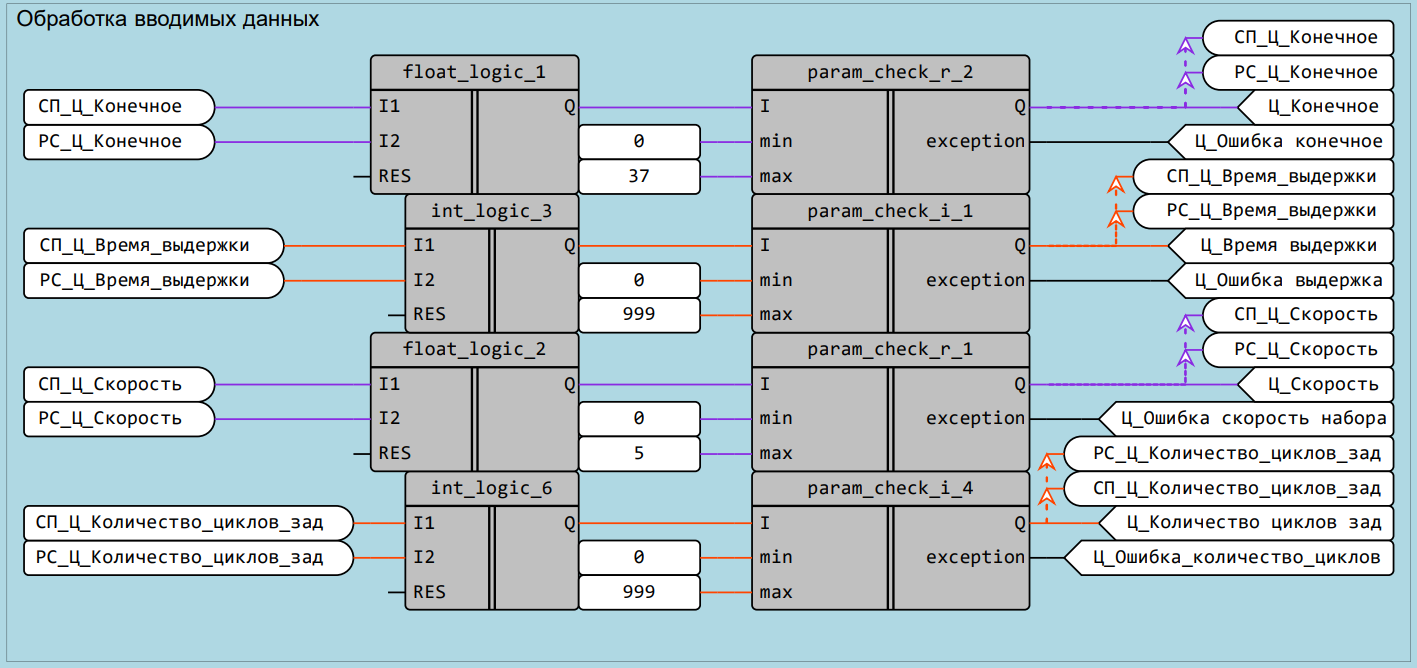


Рис. .. Блок обработки введенных значений в циклическом режиме.

Последним блоком данного режима является блок управления (см. рис.4.37), он состоит из двух функций – первая control\_cyclical\_ (код приведен в листинге П5.6), который непосредственно задает текущие требования к работе ПЧВ т.е. отвечает за выполнение самого режима, вторая же – regulation\_, которая отвечает за управление работой ПЧВ с целью достижения заданной скорости изменения давления.

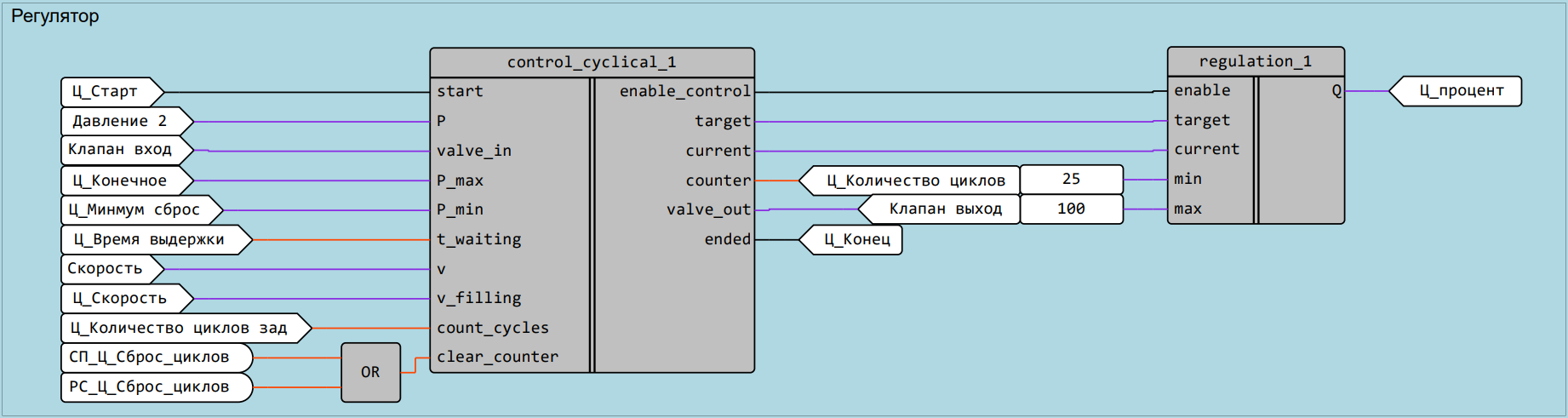


Рис.4.37. Блок управления в циклическом режиме.

В основе работы блока control\_cyclical\_ лежит конечный автомат, который приведен на рис.4.38. Как видно, он состоит из 5 состояний, и они полностью описывают алгоритм, приведенный в разделе 3.3.

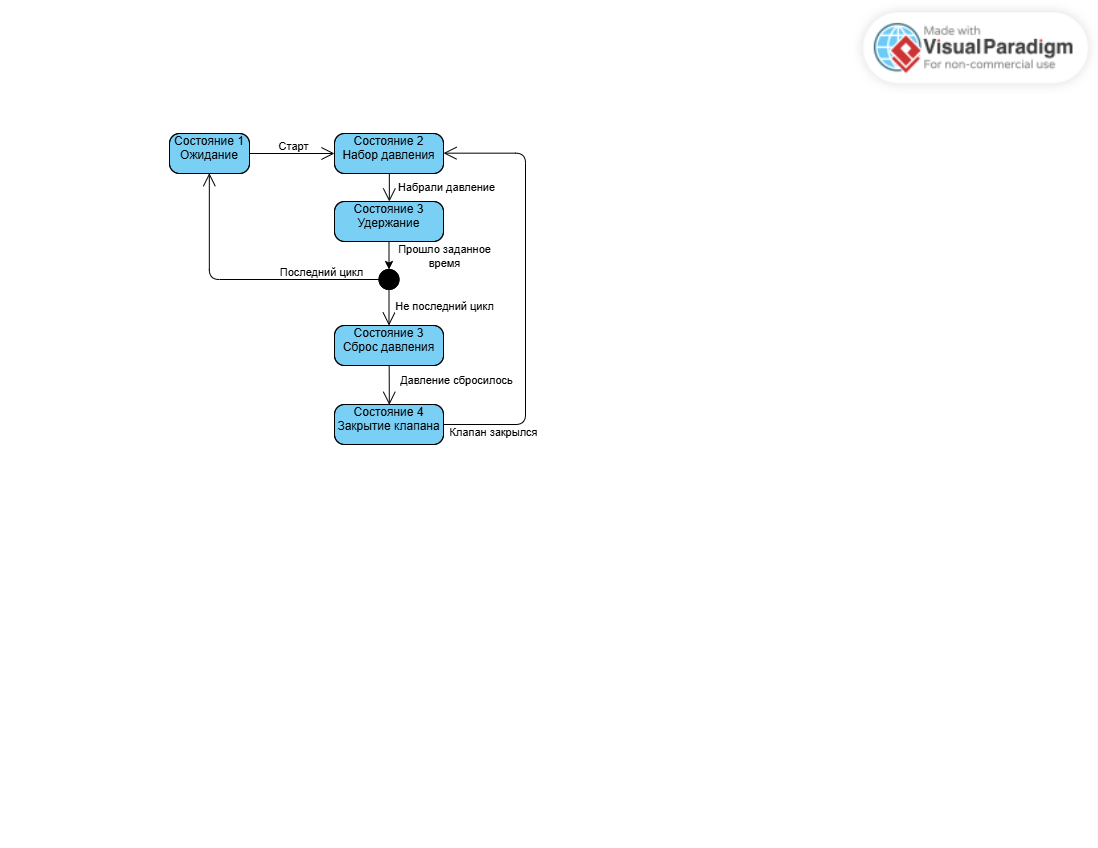


Рис.4.38. Конечный автомат для control\_cyclical\_.

В первом состоянии алгоритм ожидает команды старта после чего переходит во второе состояние. Во втором состоянии выдается команда на набор давления с заданной скоростью, после чего ожидается достижение требуемой величины, и система переходит в третье состояние. В третьем состоянии управление выключается и запускается ожидание на заданное время, после этого идет переход в четвертое состояние, если это не последний цикл и в первое, если последний. В четвертом состоянии происходит сброс давления до заданного, после этого идет переход в четвертое состояние. В пятом состоянии ожидается закрытие клапанов и переход во второе состояние для набора давления.

Блок regulation\_ основан на классическом П-регуляторе с модификацией в виде -адаптации. Классический П-регулятор в дискретном виде имеет следующую рабочую формулу:

где – управляющий сигнал;

– пропорциональный коэффициент;

– ошибка между текущей величиной и желаемой.

Проблема данного метода состоит в подборе коэффициента , а также в появлении статической ошибки, или перерегулирования, которые возникают из-за большого или малого значения . В таком случае часто модифицируют П-регулятор до ПИД и его разновидностей, однако такой метода также требует подбора коэффициентов и работает только с заданной конфигурацией системы управления, однако в нашем случае объем гидробака может меняться (в следствии погружения в него испытываемых устройств), что значительно сказывается на скорости, а значит и на ПИД регуляторе.

В качестве альтернативы был выбран П-регулятор с -адаптацией, его преимущество в том, что коэффициент подбирается динамически по следующей формуле:

где – пропорциональный коэффициент на -ом шаге;

– скорость (коэффициент) адаптации;

– ошибка на k-ом шаге, вычисляется как разница между требуемой величиной и текущей;

– коэффициент утечки, ограничивающий рост .

действует как экспоненциальный демпфер: если ошибка длительно мала (либо шум компенсировал ошибку), коэффициент постепенно возвращается к области умеренных значений, предотвращая накопление ошибок модели. При ненулевой ошибке градиентный прирост компенсирует утечку, обеспечивая адаптацию, но уже в ограниченном динамическом диапазоне [1].

Проведем испытания, запустив насос на 30%, 65% и 100%, получим результат, приведенный на рис.4.39.



Рис.4.39. График набора давления при разных частотах работы двигателя.

Посчитаем скорость на этих отрезках и нанесем её на график (см. рис.4.40).

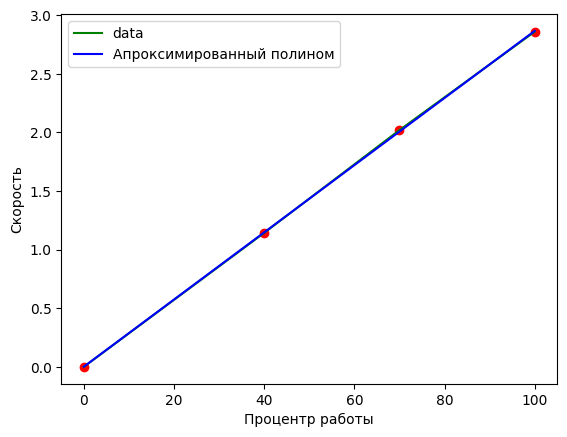


Рис.4.40. Скорость в зависимости от процента работы ПЧВ.

В итоге получилась линейная зависимость скорости от процента работы ПЧВ. Аппроксимированный полином имеет следующую формулу:

Первое приближение, с которого начинает работать ПЧВ выбирается в соответствии с формулой выше, после чего корректировкой занимается П-регулятор с -адаптацией.

В [1] рекомендуется следующий алгоритм подбора параметров для -адаптации:

Определить , для этого выполнить малое изменение управляющего сигнала и посмотреть на изменение измеряемой величины т.е. .

Далее, задать начальные переменные γ[0] = 0.05, σ[0] = 0.03.

Каждую секунду обновляем по формуле ниже:

где – пропорциональный коэффициент на -ом шаге;

– скорость (коэффициент) адаптации на -ом шаге;

– ошибка на -ом шаге (разница между требуемой величиной и текущей;

– коэффициент утечки, ограничивающий рост на -ом шаге.

Каждые 5 секунд корректируем скорость адаптации, для этого считаем среднюю относительную ошибку за N=5 секунд по этой формуле:

где – средняя относительная ошибка за последние N отсчетов;

N – количество отсчетов;

– ошибка на i-ом шаге;

R – рабочий размах измеряемой величины, возьмем её равной 5 (скорость меняется от 0 до 5 МПа/мин).

После чего обновляем значение скорости адаптации по формуле ниже:

где – скорость адаптации на k-ом шаге;

*–* средняя относительная ошибка.

А также каждые 30 секунд, во избежание слишком быстрого увеличения необходимо проверять, что , в противном случае скорость адаптации и коэффициент утечки выполняется следующим образом:

где – коэффициент утечки на k-ом шаге;

– скорость адаптации на k-ом шаге.

Этот алгоритм выполняется до тех пор, пока не станет удовлетворять требованиям проекта, в рамках СИГ было принято значение 3%, рекомендуемое как стандартное.

После завершения настройки значения скорости адаптации и утечки на последнем шаге фиксируются и в дальнейшем не изменяются в ходе работы, а пропорциональный коэффициент на последнем шаге становится новым для рабочего режима.

В ходе испытаний было получено, что необходимо взять равным 34.9, γ = 0.15, σ = 0.035. В такой конфигурации были получены наилучшие результаты. Код функции regulation\_ приведен в листинге П5.7.

Последним оставшимся режимом работы является статический режим. Он полностью приведен на рис.4.41. Как можно заметить, его блоки ничем не отличаются от аналогичных в предыдущих режимах.

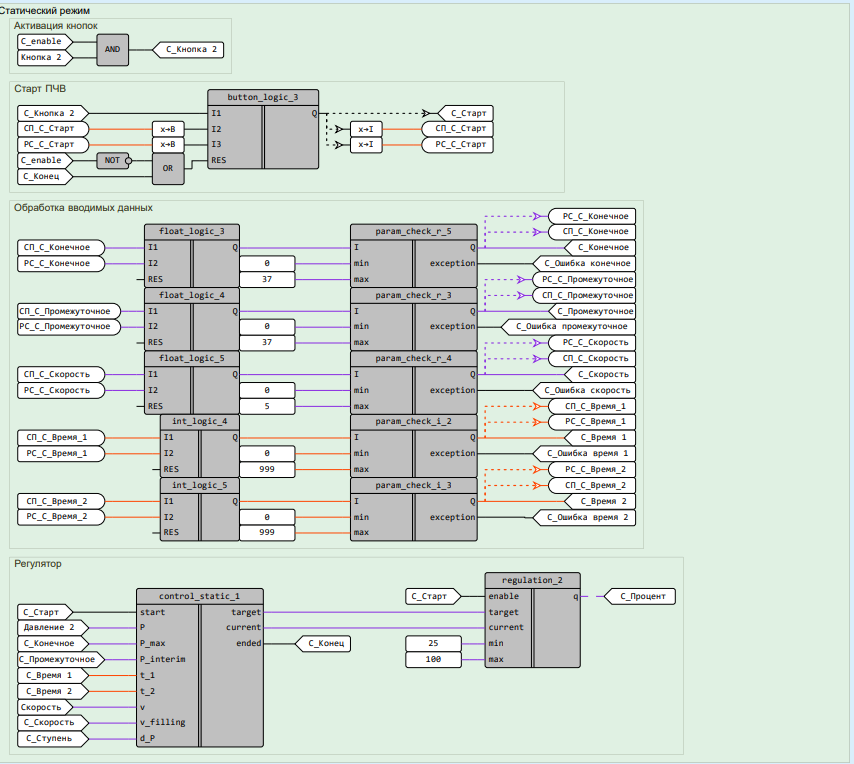


Рис.4.41. Статический режим.

Отдельно стоит рассмотреть лишь блок control\_static\_, который занимается непосредственно реализацией самого алгоритма. В его основе лежит конечный автомат, который приведен на рис.4.42.

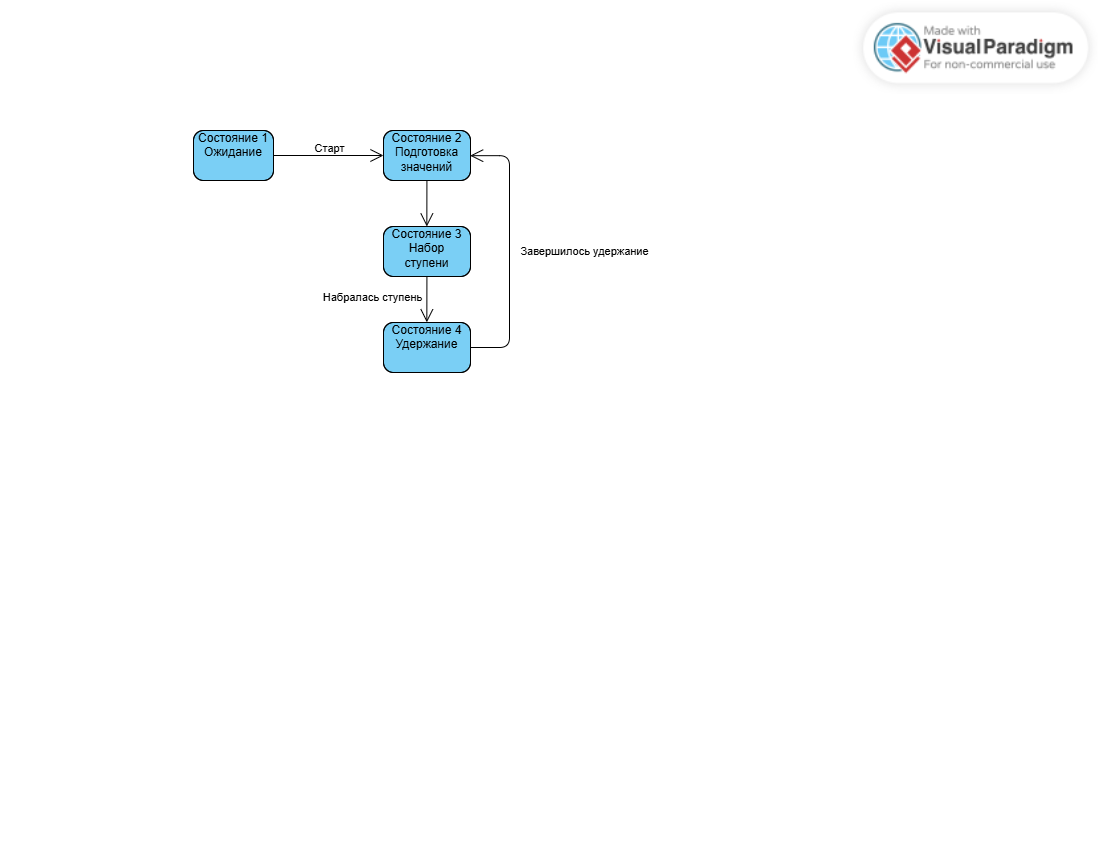


Рис.4.42. Конечный автомат блока control\_static\_.

Данный конечный автомат состоит из 4 состояний. Первое отвечает за ожидание начала работы, второй обновляет данные для следующей ступени, а четвертый выполняет удержание ступени заданное время. Код находится в листинге П5.8.

## Разработка приложения для отображения графиков после испытаний

Последнее приложение, которое необходимо разработать – генератор графиков по данным от СИГа.

На данный момент результат работы можно посмотреть либо в процессе работы на экранах СП310 или удаленном рабочем месте оператора, однако результат испытаний необходимо сохранять и приводить в отчетах. Для этого СП310 выполняет запись на USB (см. рис.4.18), однако сохраняются исключительно цифры в формате .csv. Для преобразования их в удобные для восприятия графики необходимо разработать приложение.

Как и приложение для удаленного рабочего места оператора, приложение для отрисовки графиков будет написано на языке python версии 3.12 и более, с использованием библиотеки tkinter. Для отрисовки графиков будет использоваться библиотека plotly т.к. она более дружелюбна к пользователю приложения, нежили matplotlib.

Таблица, сохраненная на флешку имеет следующие столбцы:

1. Pressure 1 – давление с первого датчика в МПа.
2. Pressure 2 – давление со второго датчика в МПа.
3. Date –дата, в формате число месяц, год, в которое было выполнено сохранение по внутренним часам панели оператора.
4. Time – время, в которое было выполнено сохранение по внутренним часам панели оператора.

Эти данные необходимо преобразовать в удобные, для отрисовки.

Для преобразования используется функция, приведенная на листинге 4.3.

Листинг 4.3. Код функции \_read\_sig\_csv.

|  |
| --- |
| **def** \_read\_sig\_csv**(**sig\_path**:** Union**[str,** Path**])** **->** Optional**[**pd**.**DataFrame**]:**  path **=** Path**(**sig\_path**)**  **if** path**.**suffix**.**lower**()** **!=** CSV\_SUFFIX**:**  **return** **None**  df **=** pd**.**read\_csv**(**path**,** delimiter**=**";"**,** decimal**=**"."**)**  # Объединяем Date и Time и преобразуем в datetime  df**[**"Time"**]** **=** pd**.**to\_datetime**(**  df**[**"Date"**]** **+** " " **+** df**[**"Time"**],** errors**=**"coerce"  **)**  df**.**drop**(**columns**=[**"Date"**],** inplace**=True)**  **return** df |

Данная функция загружает файл по переданному пути, после чего возвращает данные в удобной форме для отрисовки.

После чтения файла необходимо его отобразить, для этого используется функция с листинга 4.4.

Листинг 4.4. Код функции \_create\_pressure\_plot.

|  |
| --- |
| **def** \_create\_pressure\_plot**(**df**:** pd**.**DataFrame**,** test\_name**:** **str)** **->** go**.**Figure**:**  fig **=** go**.**Figure**()**  sensors **=** **[**  **(**"Pressure 2"**,** "Давление в гидробаке (ПД100) Д2"**),**  **(**"Pressure 1"**,** "Давление в гидробаке (ПД100) Д1"**),**  **]**  **for** column**,** label **in** sensors**:**  fig**.**add\_trace**(**  go**.**Scatter**(**  x**=**df**[**"Time"**].**iloc**[::**MARKER\_INTERVAL**],**  y**=**df**[**column**].**iloc**[::**MARKER\_INTERVAL**],**  mode**=**"lines+markers"**,**  name**=**label**,**  line**=dict(**width**=**1**),**  marker**=dict(**size**=**4**),**  **)**  **)**  **...**  **return** fig |

В приведенном выше коде опущен момент с настройкой оформления, полный код приведен на github в [9].

Как можно заметить, сама функция добавляет на график все точки с заданным в настройках шагом MARKER\_INTERVAL. Это сделано для того, чтоб не отрисовывать слишком много точек т.к. испытания могут быть долгими, это будет выглядеть плохо. После чего она возвращает настроенный объект графика.

Далее необходимо запустить листинг 4.3 и листинг 4.4 вместе, для этого используется функция, приведённая на листинге 4.5.

Листинг 4.5. Код функции plot\_sig\_data.

|  |
| --- |
| **def** plot\_sig\_data**(**sig\_path**:** Union**[str,** Path**],** test\_name**:** **str** **=** ""**)** **->** **None:**  df **=** \_read\_sig\_csv**(**sig\_path**)**  **if** df **is** **None:**  **raise** **ValueError(**f"'{sig\_path}' должен иметь расширение {CSV\_SUFFIX}"**)**  safe\_name **=** test\_name**.**replace**(**"#"**,** "№"**)**  fig **=** \_create\_pressure\_plot**(**df**,** safe\_name**)**  fig**.**show**()** |

Итого готов код для полной отрисовки графика испытания с его названием, однако вызывать код из консоли может быть не удобно и недоступно рядовому пользователю, поэтому необходимо сделать графический интерфейс, который бы позволял задавать название и путь до файла испытания.

Воспользуемся библиотекой tkinter и разработаем класс, приведенный на рис.4.43. Его код можно найти на github в [9].

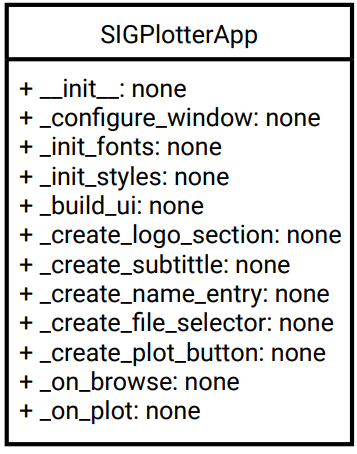


Рис.4.43. Описание класса SIGPlotterApp.

Данный класс, как понятно из названия, полностью занимается отрисовкой интерфейса, итоговый результат приведен на рис.4.44.

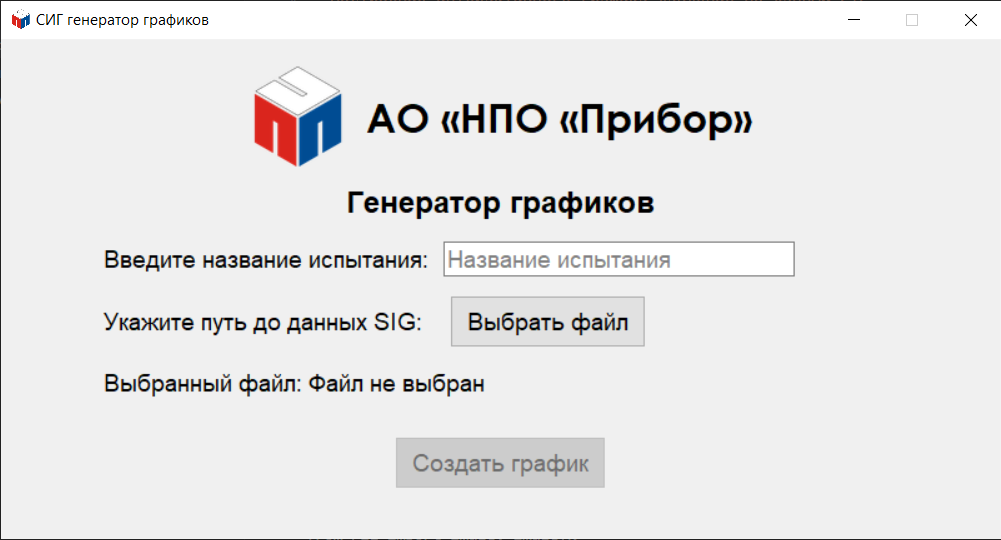


Рис.4.44. Внешний вид приложения для генерации графиков.

Все функции в классе SIGPlotterApp являются вспомогательными и их назначение понятно из названия:

* с префиксом init, занимаются инициализацией различных компонентов;
* \_configure\_window задает размеры приложения и его иконку;
* с префиксом create занимаются отрисовкой самого интерфейса;
* \_build\_ui вызывает все функции create в заданном порядке;
* с префиксом on являются функциями обратного вызова для кнопок интерфейса.

Отдельный интерес вызывает поле ввода названия. При нажатии на него оно автоматически заполняется текстом «Испытание», это сделано для того, чтоб оператору не пришлось вводить его вручную, но, при необходимости, эту фразу можно стереть. Для этого разработан отдельный класс, наследующийся от ttk.Entry, который является стандартным классом для поля ввода. Структура разработанного класса приведена на рис.4.45.

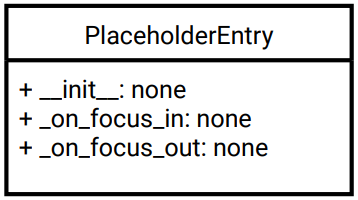


Рис.4.45. Описание класса PlaceholderEntry.

Создадим файл в стиле СИГ, со случайными данными и попытаемся его отрисовать, используя созданное приложение. Выберем файл с данными и зададим название «Испытание тест», интерфейс с заполненными полями приведен на рис.4.46.

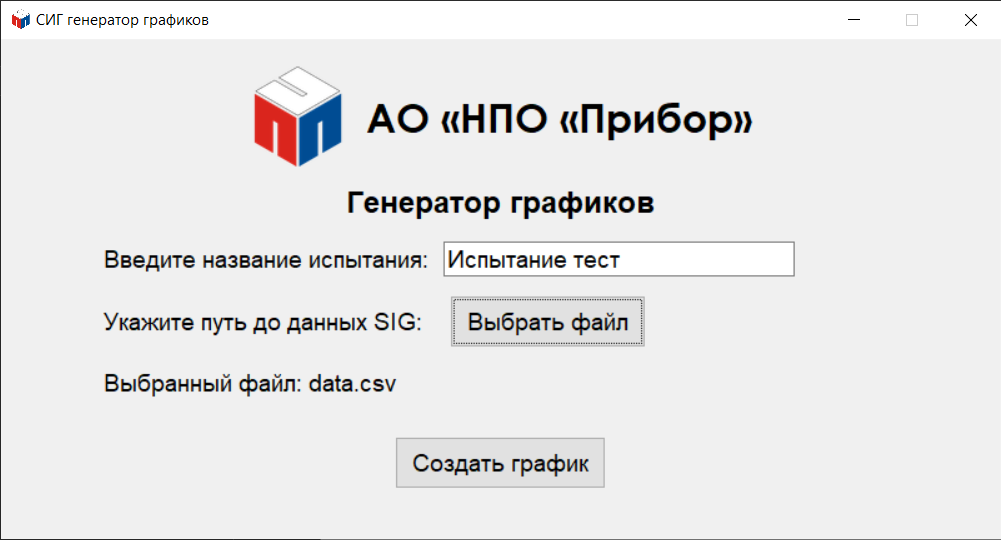


Рис.4.46. Интерфейс генератора графиков с данными.

При нажатии кнопки «Создать график» открывается браузер по умолчанию с сгенерированным графиком. Получившийся результат приведен на рис.4.47.

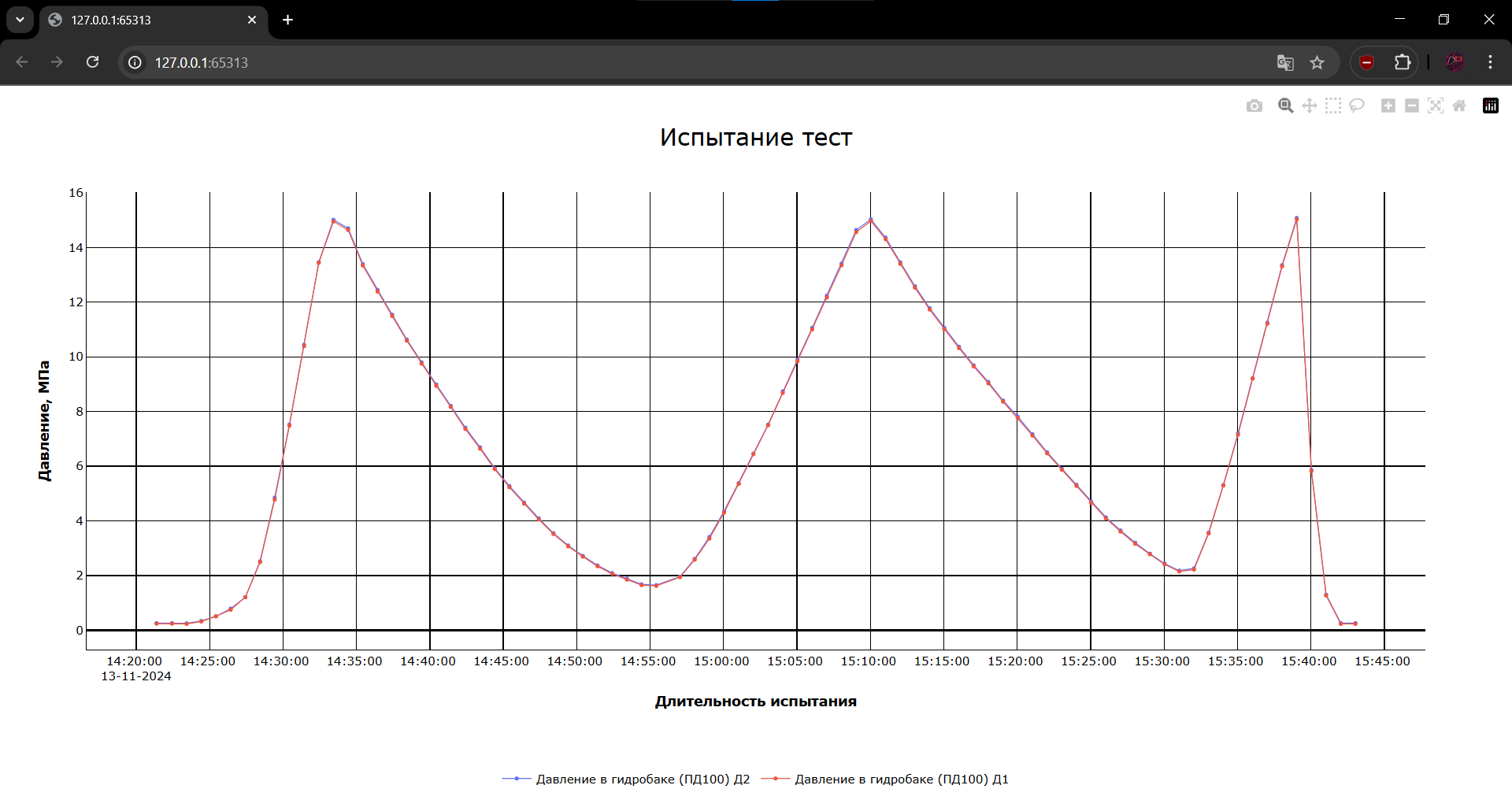


Рис.4.47. График, созданный при помощи генератора.

Таким образом приложение готово к эксплуатации. Полный код приведен на github в [9].

## Выводы по разделу

В ходе данного раздела было выполнено подключение периферийных устройств к ПР200, разработано программное обеспечение для СП310, ПР200 и удаленного рабочего места оператора. На базе ПР200 были реализованы статический и циклический режимы, а также ручной режим, которые были описаны в разделе 3.

В заключительном разделе 4.5 было разработано приложение для отображения графиков после испытаний и проверено на корректность работы на случайных данных.

# ТЕСТИРОВАНИЕ УСТАНОВКИ СИГ

## Тестирование

## Выводы по разделу

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Åström K. J., Wittenmark B. Adaptive control. - 2 изд. - Addison-Wesley, 2008. - 592 с.
2. Hydrofab // Готовые гидравлические стенды URL: https://hydrofab.ru/container-testing/ (дата обращения: 06.01.2025).
3. Lawrence M. Thompson Industrial data communications. - 4 изд. - Research Triangle Park, NC : ISA, 2008. - 268 с.
4. Savitzky A., Golay M.J.E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures // Analytical Chemistry. - 1964. - №8. - С. 1627-1639.
5. АО "НПО "Прибор" // Официальный сайт URL: https://npo-pribor.ru/ (дата обращения: 06.01.2025).
6. Барбасова Т.А., Канашев Е.А. Промышленные сети и системы связи: учебное пособие. - Челябинск: Издательский центр ЮУрГУ, 2020. - 144 с.
7. Денисенко В.В. Компьютерное управление технологическим процессом, экспериментом, оборудованием. - М.: Горячая линия - Телеком, 2009. - 608 с.
8. Елизаров И. А., Назаров В. Н., Третьяков А. А. Промышленные вычислительные сети: учебное пособие. - Тамбов: Издательский центр ФГБОУ ВО «ТГТУ», 2024. - 98 с.
9. Исходный код программ // Github URL: https://github.com/DafterT/FQW-SIG/tree/main/SIG (дата обращения: 26.05.2025).
10. Паспорт ЭПК BVM // Компания BVM URL: https://bvm-privod.ru/upload/iblock/eae/sdd65d4i5bglmau4i2mrr4itq9mwr54k/%D0%9F%D0%90%D0%A1%D0%9F%D0%9E%D0%A0%D0%A2%20%D0%AD%D0%9F%D0%9A%20BVM.pdf (дата обращения: 18.04.2025).
11. ПД100 датчик преобразователь избыточного давления // Компания ОВЕН URL: https://owen.ru/product/datchik\_preobrazovatel\_izbitochnogo\_davleniya\_PD100\_dlya\_nasosov\_kotelnykh\_vodosnabzheniya (дата обращения: 06.01.2025).
12. Плунжерный дозировочный насос НД 25/400 К14А // Компания АРЕОПАГ URL: https://areopag-spb.ru/pumps/dozirovochnye\_plunzhernye/dozirovochnye\_nasosy/nd\_2\_5\_25\_400\_k14a\_v/ (дата обращения: 06.01.2025).
13. ПР200 программируемое реле // Компания ОВЕН URL: https://owen.ru/product/pr200 (дата обращения: 06.01.2025).
14. ПЧВ1 частотный преобразователь // Компания ОВЕН URL: https://owen.ru/product/pchv\_m01 (дата обращения: 06.01.2025).
15. СП3хх сенсорные панели оператора // Компания ОВЕН URL: https://owen.ru/product/sp3xx (дата обращения: 06.01.2025).
16. Схема щита СИГ // Github URL: https://github.com/DafterT/FQW-SIG/tree/main/Report/5.%20%D0%92%D0%9A%D0%A0/%D0%A9%D0%B8%D1%82 (дата обращения: 26.05.2025).
17. ЦНИИ "Электроприбор" // Официальный сайт URL: http://www.elektropribor.spb.ru (дата обращения: 06.01.2025).
18. Электропривод BVM ЭПК24АВ // Компания BVM URL: https://bvm-privod.ru/product/avtomatika/krany-s-elektroprivodami/elektroprivody-dlya-upravleniya-smesitelnymi-klapanami/epk24av/ (дата обращения: 16.04.2025).

# Приложение 1

**ТАБЛИЦА РЕГИСТРОВ ДЛЯ УПРАВЛЕНИЯ ПЧВ**

Таблица П1.

Регистры управления ПЧВ1

|  |  |  |  |
| --- | --- | --- | --- |
| Регистр (hex) | Назначение | Ед. изм. / диапазон | Описание и возможные значения |
| 0x2000 | Заданная частота. | 0.01 Гц (0.00 … 320.00 Гц) | Выражается в сотых герца. |
| 0x2001 | Команда управления. | 0x0000…0x0103 | **Бит 0 (0x0001)** – «Пуск прямое»: при 1 – команда на запуск двигателя в прямом направлении.  **Бит 1 (0x0002)** – «Пуск обратное»: при 1 – команда на запуск двигателя в обратном направлении.  **Бит 2 (0x0004)** – «Стоп с замедлением»: при 1 – останов двигателя по профилю замедления.  **Бит 3 (0x0008)** – «Стоп экстренный»: при 1 – мгновенная остановка без учёта профиля.  **Бит 4 (0x0010)** – «Сброс аварии»: при 1 – сброс флага аварийного состояния.  **Бит 5 (0x0020)** – «Блокировка запуска»: при 1 – блокирует любые команды запуска до разблокировки.  **Бит 6 (0x0040)** – «Разблокировка запуска»: при 1 – снимает блокировку, позволяет последующие запуски.  **Бит 7 (0x0080)** – «Сохранённая команда»: при 1 – индикация наличия незавершённой команды (внутренний флаг).  **Биты 8…15 (0x0100…0x8000)** – зарезервированы и должны оставаться равными 0. |

# Приложение 2

**Экраны для панели оператора**

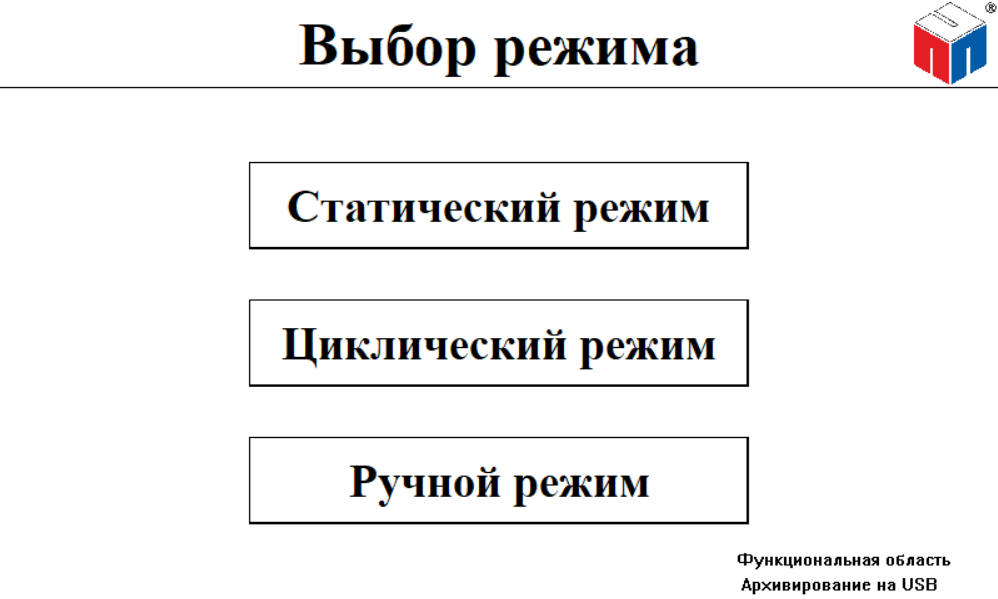
****

Рис.П2.1. Экран выбора режима.

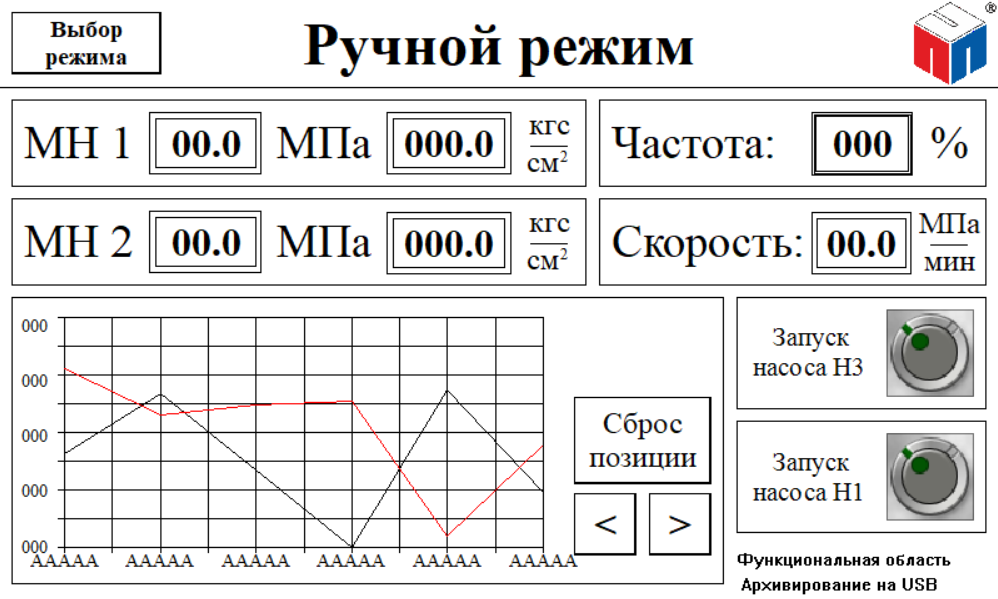
****

Рис.П2.2. Экран управления ручным режимом.

****

Рис.П2.3. Экран настроек циклического режима.



Рис.П2.4. Экран управления циклическим режимом.

****

Рис.П2.5. Экран настроек статического режима.

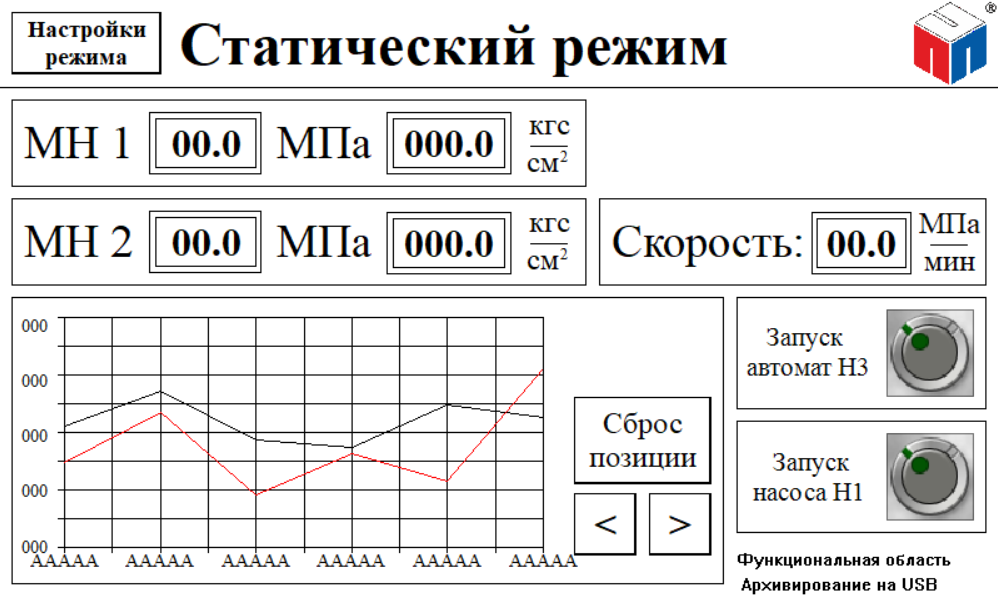
****

Рис.П2.6. Экран управления статическим режимом.

# Приложение 3

**Листинг приложения оператора для компьютера**

Листинг П3.1. Файл main.py

|  |
| --- |
| **from** App **import** App  **if** \_\_name\_\_ **==** "\_\_main\_\_"**:**  app **=** App**()**  app**.**mainloop**()** |

Листинг П3.2. Файл App.py

|  |
| --- |
| **import** tkinter **as** tk  **from** tkinter **import** ttk  **from** frames**.**MainMenu **import** MainMenu  **from** frames**.**StatSettings **import** StatSettings  **from** frames**.**CycleSettings **import** CycleSettings  **from** frames**.**ManualMode **import** ManualMode  **from** frames**.**StatMode **import** StatMode  **from** frames**.**CycleMode **import** CycleMode  **from** utils**.**ModbusSlave **import** ModbusSlave  **from** utils**.**constants\_for\_regs **import** **\***  **import** bidict  **from** utils**.**math\_functions **import** get\_kgs  **class** **App(**tk**.**Tk**):**  **def** \_\_init\_\_**(**self**):**  **super().**\_\_init\_\_**()**  self**.**title**(**"Управление режимами"**)**  self**.**geometry**(**"800x480"**)**  self**.**screen\_numbers **=** bidict**.**bidict**({**  "MainMenu"**:** 3**,**  "ManualMode"**:** 4**,**  "CycleMode"**:** 5**,**  "StatMode"**:** 6**,**  "StatSettings"**:** 7**,**  "CycleSettings"**:** 8**,**  **})**  # Контейнер для всех фреймов  self**.**container **=** tk**.**Frame**(**self**)**  self**.**container**.**pack**(**side**=**"top"**,** fill**=**"both"**,** expand**=True)**  self**.**container**.**grid\_rowconfigure**(**0**,** weight**=**1**)**  self**.**container**.**grid\_columnconfigure**(**0**,** weight**=**1**)**  self**.**after\_id **=** **None**  self**.**plots\_upd\_id **=** **None**  self**.**style **=** ttk**.**Style**()**  self**.**style**.**theme\_use**(**'alt'**)**  self**.**current\_frame **=** **None** # Текущий активный фрейм  # Установка шрифта по умолчанию для всех кнопок  self**.**style**.**configure**(**"Main.TButton"**,**  font**=(**'Times New Roman'**,** 32**),**  padding**=**6**,** anchor**=**"center"**,** background**=**"white"**)**  self**.**style**.**configure**(**"Second.TButton"**,**  font**=(**'Times New Roman'**,** 14**),**  padding**=**0**,** anchor**=**"nsew"**,** wraplength**=**120**,** background**=**"white"**)**  self**.**style**.**configure**(**"Main.TEntry"**,**  padding**=**1**,** foreground**=**"black"**)**  self**.**style**.**configure**(**"Main.TLabel"**,**  font**=(**'Times New Roman'**,** 22**),**  padding**=**0**,** background**=**"white"**,** anchor**=**"center"**,** borderwidth**=**3**,** bordercolor**=**"black"**,**  relief**=**"solid"**)**  self**.**style**.map(**"NonActive.TButton"**,**  background**=[(**"!active"**,** "#808080"**),** **(**"active"**,** "#b3b3b3"**)],**  foreground**=[(**"!active"**,** "white"**),** **(**"active"**,** "white"**)])**  self**.**style**.**configure**(**"NonActive.TButton"**,**  font**=(**'Times New Roman'**,** 14**),**  wraplength**=**120**)**  self**.**style**.map(**"Active.TButton"**,**  background**=[(**"!active"**,** "#32cd32"**),** **(**"active"**,** "#5ad65a"**)],**  foreground**=[(**"!active"**,** "black"**),** **(**"active"**,** "black"**)])**  self**.**style**.**configure**(**"Active.TButton"**,**  font**=(**'Times New Roman'**,** 14**),**  wraplength**=**120**)**  self**.**speed\_mpa\_var **=** tk**.**DoubleVar**(**value**=**3.2**)**  self**.**mn1\_mpa\_var **=** tk**.**DoubleVar**()**  self**.**mn1\_kgs\_var **=** tk**.**DoubleVar**()**  self**.**mn1\_mpa\_var**.**trace\_add**(**"write"**,** self**.**update\_kgs1**)**  self**.**mn1\_mpa\_var**.set(**0.2**)**  self**.**mn2\_mpa\_var **=** tk**.**DoubleVar**()**  self**.**mn2\_kgs\_var **=** tk**.**DoubleVar**()**  self**.**mn2\_mpa\_var**.**trace\_add**(**"write"**,** self**.**update\_kgs2**)**  self**.**mn2\_mpa\_var**.set(**0.3**)**  self**.**number\_of\_cycles\_var **=** tk**.**IntVar**()**  self**.**number\_of\_cycles\_var**.set(**0**)**  self**.**frames **=** **{}**  self**.**slave **=** ModbusSlave**(**baudrate**=**38400**,** slave\_id**=**2**)**  self**.**slave**.**set\_callback**(**self**.**slave**.**WRITE\_MULTIPLE\_REGISTERS**,** self**.**write\_registers\_callback**)**  self**.**slave**.**set\_callback**(**self**.**slave**.**WRITE\_SINGLE\_REGISTER**,** self**.**write\_registers\_callback**)**  # Создаем все экраны  **for** F **in** **(**MainMenu**,** StatSettings**,** CycleSettings**,** ManualMode**,** StatMode**,** CycleMode**):**  frame **=** F**(**self**.**container**,** self**)**  self**.**frames**[**F**.**\_\_name\_\_**]** **=** frame  frame**.**grid**(**row**=**0**,** column**=**0**,** sticky**=**"nsew"**)**  self**.**show\_frame**(**"MainMenu"**)**    **try:**  self**.**slave**.**start**()**  **except** **Exception** **as** e**:**  **print(**e**)**  self**.**protocol**(**"WM\_DELETE\_WINDOW"**,** self**.**on\_close**)**  self**.**\_center\_window**()**  self**.**plots\_upd\_id **=** self**.**after**(**1000**,** self**.**plots\_upd**)**  **def** plots\_upd**(**self**):**  pressure **=** self**.**frames**[**"ManualMode"**].**get\_float\_from\_registers**(**PRESSURE\_MN1**)**  self**.**frames**[**"ManualMode"**].**pressure\_graph**.**update\_data**(**pressure**)**  self**.**frames**[**"StatMode"**].**pressure\_graph**.**update\_data**(**pressure**)**  self**.**frames**[**"CycleMode"**].**pressure\_graph**.**update\_data**(**pressure**)**  # UPD  self**.**plots\_upd\_id **=** self**.**after**(**1000**,** self**.**plots\_upd**)**  **def** on\_close**(**self**):**  **for** frame **in** self**.**frames**.**values**():**  frame**.**event\_generate**(**"<<HideFrame>>"**)**  **try:**  self**.**slave**.**stop**()**  **except** **Exception** **as** e**:**  **print(**e**)**  self**.**frames**[**"ManualMode"**].**pressure\_graph**.**cleanup**()** **if** **hasattr(**self**.**frames**[**"ManualMode"**],**  'pressure\_graph'**)** **else** **None**  self**.**frames**[**"StatMode"**].**pressure\_graph**.**cleanup**()** **if** **hasattr(**self**.**frames**[**"StatMode"**],**  'pressure\_graph'**)** **else** **None**  self**.**frames**[**"CycleMode"**].**pressure\_graph**.**cleanup**()** **if** **hasattr(**self**.**frames**[**"CycleMode"**],**  'pressure\_graph'**)** **else** **None**  self**.**destroy**()**  **def** write\_registers\_callback**(**self**,** request**):**  # Обновление экрана  **if** self**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **!=** self**.**screen\_numbers**[**self**.**current\_frame**.**\_\_class\_\_**.**\_\_name\_\_**]:**  self**.**show\_frame**(**self**.**screen\_numbers**.**inverse**[**self**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]])**    # Обновление состояния кнопок  self**.**frames**[**"ManualMode"**].**update\_button\_state\_by\_register**(**START\_AUTOMAT\_N3\_MANUAL\_REG**,** START\_MODE\_MANUAL\_REG**)**  self**.**frames**[**"StatMode"**].**update\_button\_state\_by\_register**(**START\_AUTOMAT\_N3\_STAT\_REG**,** START\_MODE\_STAT\_REG**)**  self**.**frames**[**"CycleMode"**].**update\_button\_state\_by\_register**(**START\_AUTOMAT\_N3\_CYCLE\_REG**,** START\_MODE\_CYCLE\_REG**)**  **def** show\_frame**(**self**,** cont**):**  """Показ фрейма с обработкой переключения"""  # Уведомляем предыдущий фрейм о скрытии  **if** self**.**current\_frame**:**  self**.**current\_frame**.**event\_generate**(**"<<HideFrame>>"**)**  # Получаем новый фрейм  frame **=** self**.**frames**[**cont**]**  # Уведомляем новый фрейм о показе  frame**.**event\_generate**(**"<<ShowFrame>>"**)**  frame**.**tkraise**()**  # Обновляем текущий фрейм  self**.**current\_frame **=** frame  **def** \_center\_window**(**self**):**  self**.**update\_idletasks**()**  width **=** self**.**winfo\_width**()**  height **=** self**.**winfo\_height**()**  x **=** **(**self**.**winfo\_screenwidth**()** **//** 2**)** **-** **(**width **//** 2**)**  y **=** **(**self**.**winfo\_screenheight**()** **//** 2**)** **-** **(**height **//** 2**)**  self**.**geometry**(**f'{width}x{height}+{x}+{y}'**)**    **def** update\_kgs1**(**self**,** **\***args**):**  mpa\_value **=** self**.**mn1\_mpa\_var**.**get**()**  rounded\_kgs **=** **round(**get\_kgs**(**mpa\_value**),** 1**)**  self**.**mn1\_kgs\_var**.set(**rounded\_kgs**)**  **def** update\_kgs2**(**self**,** **\***args**):**  mpa\_value **=** self**.**mn2\_mpa\_var**.**get**()**  rounded\_kgs **=** **round(**get\_kgs**(**mpa\_value**),** 1**)**  self**.**mn2\_kgs\_var**.set(**rounded\_kgs**)** |

Листинг П3.3. Файл ModbusSlave.py

|  |
| --- |
| **import** struct  **import** threading  **import** serial  **import** serial**.**tools**.**list\_ports  **import** time  **from** crcmod **import** mkCrcFun  **class** **ModbusSlave:**  """  Класс реализующий Modbus Slave (сервер)  с возможностью установки колбэков при получении корректных запросов.  """  # Modbus исключения  ILLEGAL\_FUNCTION **=** 0x01  ILLEGAL\_DATA\_ADDRESS **=** 0x02  ILLEGAL\_DATA\_VALUE **=** 0x03  # Функции Modbus  READ\_HOLDING\_REGISTERS **=** 0x03  WRITE\_SINGLE\_REGISTER **=** 0x06  WRITE\_MULTIPLE\_REGISTERS **=** 0x10  **def** \_\_init\_\_**(**  self**,** baudrate**=**9600**,** timeout**=**1**,** slave\_id**=**1**,** bytesize**=**8**,** parity**=**"E"**,** stopbits**=**1  **):**  """  Инициализация Modbus Slave  :param baudrate: скорость для RTU  :param timeout: таймаут ожидания данных  :param slave\_id: идентификатор устройства (1-247)  """  self**.**thread **=** **None**  self**.**port **=** **None**  self**.**baudrate **=** baudrate  self**.**timeout **=** timeout  self**.**bytesize **=** bytesize  self**.**parity **=** parity  self**.**stopbits **=** stopbits  self**.**serial **=** serial**.**Serial**()**  self**.**slave\_id **=** slave\_id  self**.**running **=** **False**  self**.**callbacks **=** **{}**  self**.**data\_store **=** **{**  "holding\_registers"**:** **[**0**]** **\*** 256**,**  **}**  # Инициализация CRC функции для RTU  self**.**crc16 **=** mkCrcFun**(**0x18005**,** rev**=True,** initCrc**=**0xFFFF**,** xorOut**=**0x0000**)**  self**.**command\_list **=** **[**self**.**WRITE\_SINGLE\_REGISTER**,** self**.**WRITE\_MULTIPLE\_REGISTERS**,** self**.**READ\_HOLDING\_REGISTERS**]**  **def** start**(**self**):**  """Запуск сервера Modbus Slave с автоопределением порта"""  self**.**running **=** **True**  **if** **hasattr(**self**,** "serial"**)** **and** self**.**serial**.**is\_open**:**  self**.**serial**.**close**()**  # Попытка автоматического определения порта  **if** self**.**\_auto\_detect\_port**():**  self**.**thread **=** threading**.**Thread**(**target**=**self**.**\_rtu\_loop**)**  self**.**thread**.**daemon **=** **True**  self**.**thread**.**start**()**  **else:**  **print(**"Не удалось найти подходящий COM-порт"**)**  self**.**running **=** **False**  **return**  **def** \_auto\_detect\_port**(**self**,** timeout**=**2.0**):**  """Автоматическое определение COM-порта"""  # Получаем список доступных портов  available\_ports **=** serial**.**tools**.**list\_ports**.**comports**()**  test\_ports **=** **[**p**.**device **for** p **in** available\_ports**]**  **if** **not** test\_ports**:**  **print(**"Нет доступных COM-портов"**)**  **return** **False**  **print(**f"Доступные порты: {test\_ports}"**)**  **for** port **in** test\_ports**:**  **try:**  **print(**f"Проверка порта {port}..."**)**  ser **=** serial**.**Serial**(**  port**=**port**,**  baudrate**=**self**.**baudrate**,**  bytesize**=**self**.**bytesize**,**  parity**=**self**.**parity**,**  stopbits**=**self**.**stopbits**,**  timeout**=**self**.**timeout**,**  **)**  # Проверяем, есть ли входящие данные  start\_time **=** time**.**time**()**  data\_received **=** **False**  **while** time**.**time**()** **-** start\_time **<** timeout**:**  **if** ser**.**in\_waiting **>** 0**:**  data **=** ser**.**read**(**ser**.**in\_waiting**)**  **print(**f"Получены данные на порту {port}: {data**.hex()**}"**)**  data\_received **=** **True**  **break**  time**.**sleep**(**0.1**)**  **if** data\_received**:**  **print(**f"Выбран порт {port} (обнаружена активность)"**)**  self**.**port **=** port  self**.**serial **=** ser  **return** **True**  **else:**  **print(**f"Порт {port} доступен, но активность не обнаружена"**)**  ser**.**close**()**  **except** serial**.**SerialException **as** e**:**  **print(**f"Ошибка при работе с портом {port}: {**str(**e**)**}"**)**  **continue**  # Если ни один порт не подошел, пробуем использовать указанный в конфигурации  **try:**  **print(**f"Попытка использовать указанный порт {self**.**port}"**)**  self**.**serial **=** serial**.**Serial**(**  port**=**self**.**port**,**  baudrate**=**self**.**baudrate**,**  bytesize**=**self**.**bytesize**,**  parity**=**self**.**parity**,**  stopbits**=**self**.**stopbits**,**  timeout**=**self**.**timeout**,**  **)**  **return** **True**  **except** serial**.**SerialException **as** e**:**  **print(**f"Не удалось открыть указанный порт {self**.**port}: {**str(**e**)**}"**)**  **return** **False**  **def** stop**(**self**):**  """Остановка сервера Modbus Slave"""  **if** self**.**running**:**  self**.**running **=** **False**  self**.**thread**.**join**()**  **if** self**.**serial**.**is\_open**:**  self**.**serial**.**close**()**  **def** set\_callback**(**self**,** function\_code**,** callback**):**  """  Установка колбэка для определенной функции Modbus  :param function\_code: код функции Modbus  :param callback: функция обратного вызова (принимает данные запроса)  """  self**.**callbacks**[**function\_code**]** **=** callback  **def** \_rtu\_loop**(**self**):**  """Основной цикл обработки запросов в режиме RTU"""  buffer **=** **bytearray()**  **while** self**.**running**:**  # Чтение данных из последовательного порта  data **=** self**.**serial**.**read**(**1**)**  **if** **not** data**:**  **continue**  buffer**.**extend**(**data**)**  **if** **len(**buffer**)** **==** 1**:**  **if** buffer**[**0**]** **!=** self**.**slave\_id**:** # 0 - широковещательный адрес  buffer**.**clear**()**  **continue**  **if** **len(**buffer**)** **==** 2**:**  **if** buffer**[**1**]** **not** **in** self**.**command\_list**:**  buffer**.**clear**()**  **continue**  # Проверка на полный пакет  expected\_length **=** self**.**\_get\_expected\_rtu\_length**(**buffer**)**  **if** **len(**buffer**)** **<** expected\_length**:**  **continue**  # Проверка CRC  crc\_received **=** buffer**[-**2**:]** # Последние 2 байта - CRC  crc\_calculated **=** self**.**\_calculate\_crc**(**buffer**[:-**2**])**  **if** crc\_received **!=** crc\_calculated**:**  buffer**.**clear**()**  **continue**  # Обработка корректного пакета  request **=** buffer**.**copy**()**  **print(**"REQUEST "**,** buffer**.hex())**  buffer**.**clear**()**  # Формирование ответа  response **=** self**.**\_process\_request**(**request**)**  **if** response**:**  **print(**"RESPONSE "**,** response**.hex())**  self**.**serial**.**write**(**response**)**  # Вызов колбэка если он установлен  function\_code **=** request**[**1**]**  **if** function\_code **in** self**.**callbacks**:**  self**.**callbacks**[**function\_code**](**request**)**  **def** \_get\_expected\_rtu\_length**(**self**,** data**):**  """Определение ожидаемой длины пакета RTU на основе кода функции"""  **if** **len(**data**)** **<** 2**:**  **return** 4 # минимальная длина (адрес + функция + CRC)  # Базовые длины для разных функций  lengths **=** **{**  self**.**READ\_HOLDING\_REGISTERS**:** 8**,**  self**.**WRITE\_SINGLE\_REGISTER**:** 8**,**  self**.**WRITE\_MULTIPLE\_REGISTERS**:** 9 **+** data**[**6**]** **if** **len(**data**)** **>** 7 **else** 9**,**  **}**  function\_code **=** data**[**1**]**  **return** lengths**.**get**(**function\_code**,** 256**)** # 256 - максимальная длина по умолчанию  **def** \_calculate\_crc**(**self**,** data**):**  """Вычисление CRC16 для RTU пакета"""  crc **=** self**.**crc16**(**data**)**  **return** struct**.**pack**(**"<H"**,** crc**)**  **def** \_process\_request**(**self**,** request**):**  """Обработка Modbus запроса и формирование ответа"""  slave\_id **=** request**[**0**]**  pdu **=** request**[**1**:-**2**]** # Исключаем адрес и CRC  function\_code **=** pdu**[**0**]** **if** **len(**pdu**)** **>** 1 **else** **None**  **try:**  match function\_code**:**  case self**.**READ\_HOLDING\_REGISTERS**:**  **return** self**.**\_read\_holding\_registers**(**slave\_id**,** pdu**)**  case self**.**WRITE\_SINGLE\_REGISTER**:**  **return** self**.**\_write\_single\_register**(**slave\_id**,** pdu**)**  case self**.**WRITE\_MULTIPLE\_REGISTERS**:**  **return** self**.**\_write\_multiple\_registers**(**slave\_id**,** pdu**)**  case \_**:**  **return** self**.**\_exception\_response**(**  slave\_id**,** function\_code**,** self**.**ILLEGAL\_FUNCTION  **)**  **except** **IndexError:**  **return** self**.**\_exception\_response**(**  slave\_id**,** function\_code**,** self**.**ILLEGAL\_DATA\_ADDRESS  **)**  **except** **Exception:**  **return** self**.**\_exception\_response**(**  slave\_id**,** function\_code**,** self**.**ILLEGAL\_DATA\_VALUE  **)**  **def** \_write\_multiple\_registers**(**self**,** slave\_id**,** pdu**):**  address **=** struct**.**unpack**(**">H"**,** pdu**[**1**:**3**])[**0**]**  quantity **=** struct**.**unpack**(**">H"**,** pdu**[**3**:**5**])[**0**]**  reg\_data **=** **[]**  **for** i **in** **range(**quantity**):**  data **=** struct**.**unpack**(**">H"**,** pdu**[**6 **+** i **\*** 2 **:** 8 **+** i **\*** 2**])[**0**]**  self**.**data\_store**[**"holding\_registers"**][**address **+** i**]** **=** data  reg\_data**.**append**(**data**)**  response **=** **(**  **bytearray([**slave\_id**,** self**.**WRITE\_MULTIPLE\_REGISTERS**])** **+** pdu**[**1**:**3**]** **+** pdu**[**3**:**5**]**  **)**  response **+=** self**.**\_calculate\_crc**(**response**)**  **return** response  **def** \_write\_single\_register**(**self**,** slave\_id**,** pdu**):**  address **=** struct**.**unpack**(**">H"**,** pdu**[**1**:**3**])[**0**]**  data **=** struct**.**unpack**(**">H"**,** pdu**[**3**:**5**])[**0**]**  self**.**data\_store**[**"holding\_registers"**][**address**]** **=** data  response **=** **(**  **bytearray([**slave\_id**,** self**.**WRITE\_SINGLE\_REGISTER**])**  **+** address**.**to\_bytes**(**2**,** "big"**)**  **+** data**.**to\_bytes**(**2**,** "big"**)**  **)**  response **+=** self**.**\_calculate\_crc**(**response**)**  **return** response  **def** \_read\_holding\_registers**(**self**,** slave\_id**,** pdu**):**  """Обработка функции чтения holding регистров (0x03)"""  address **=** struct**.**unpack**(**">H"**,** pdu**[**1**:**3**])[**0**]**  quantity **=** struct**.**unpack**(**">H"**,** pdu**[**3**:**5**])[**0**]**  **if** quantity **<** 1 **or** quantity **>** 125**:**  **return** self**.**\_exception\_response**(**  slave\_id**,** self**.**READ\_HOLDING\_REGISTERS**,** self**.**ILLEGAL\_DATA\_VALUE  **)**  end\_address **=** address **+** quantity **-** 1  **if** end\_address **>=** **len(**self**.**data\_store**[**"holding\_registers"**]):**  **return** self**.**\_exception\_response**(**  slave\_id**,** self**.**READ\_HOLDING\_REGISTERS**,** self**.**ILLEGAL\_DATA\_ADDRESS  **)**  # Формирование данных ответа  byte\_count **=** quantity **\*** 2  registers\_data **=** **bytearray(**byte\_count**)**  **for** i **in** **range(**quantity**):**  registers\_data**[**i **\*** 2 **:** **(**i **\*** 2**)** **+** 2**]** **=** struct**.**pack**(**  ">H"**,** self**.**data\_store**[**"holding\_registers"**][**address **+** i**]**  **)**  # Формирование ответа  response **=** **(**  **bytearray([**slave\_id**,** self**.**READ\_HOLDING\_REGISTERS**,** byte\_count**])**  **+** registers\_data  **)**  response **+=** self**.**\_calculate\_crc**(**response**)**  **return** response  **def** \_exception\_response**(**self**,** slave\_id**,** function\_code**,** exception\_code**):**  """Формирование ответа с исключением"""  response **=** **bytearray([**slave\_id**,** function\_code **|** 0x80**,** exception\_code**])**  response **+=** self**.**\_calculate\_crc**(**response**)**  **return** response |

Листинг П3.4. Файл BaseFrame.py

|  |
| --- |
| **import** struct  **import** tkinter **as** tk  **import** os  **from** PIL **import** Image**,** ImageTk  **from** tkinter **import** simpledialog  **from** utils**.**constants\_for\_regs **import** **\***  **import** sys  **class** **BaseFrame(**tk**.**Frame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**)**  self**.**start\_mode **=** **None**  self**.**start\_automat\_n3 **=** **None**  self**.**controller **=** controller  self**.**bind**(**"<<ShowFrame>>"**,** self**.**on\_show\_frame**)**  self**.**bind**(**"<<HideFrame>>"**,** self**.**on\_hide\_frame**)**  self**.**after\_id **=** **None**  # Переменные для фона  self**.**bg\_image **=** **None**  self**.**bg\_image\_raw **=** **None**  self**.**bg\_label **=** tk**.**Label**(**self**)**  self**.**bg\_label**.**place**(**x**=**0**,** y**=**0**,** relwidth**=**1**,** relheight**=**1**)**  # Привязка к изменению размера окна  self**.**bind**(**"<Configure>"**,** self**.**\_resize\_background**)**  self**.**resources\_dir **=** os**.**path**.**join**(**os**.**path**.**dirname**(**\_\_file\_\_**),** '..'**,** 'imgs'**)**  **def** load\_image**(**self**,** filename**,** size**=None):**  """Универсальная загрузка изображений с обработкой ошибок"""  **try:**  path **=** os**.**path**.**join**(**self**.**resources\_dir**,** filename**)**  img **=** Image**.open(**path**)**  **if** size**:**  img **=** img**.**resize**(**size**,** Image**.**Resampling**.**LANCZOS**)**  **return** ImageTk**.**PhotoImage**(**img**)**  **except** **Exception** **as** e**:**  **print(**f"Ошибка загрузки {filename}: {e}"**)**  **return** **None**  **def** on\_show\_frame**(**self**,** event**=None):**  # Метод, вызываемый при показе фрейма  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "MainMenu"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 3  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "ManualMode"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 4  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "CycleMode"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 5  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "StatMode"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 6  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "StatSettings"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 7  **if** self**.**\_\_class\_\_**.**\_\_name\_\_ **==** "CycleSettings"**:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**CURRENT\_FRAME\_REG**]** **=** 8  **def** on\_hide\_frame**(**self**,** event**=None):**  """Вызывается при скрытии фрейма"""  **pass**  **def** set\_background**(**self**,** image\_path**):**  """Установка фонового изображения с автоматическим масштабированием"""  **try:**  self**.**bg\_image\_raw **=** Image**.open(**image\_path**)**  self**.**\_resize\_background**()**  **except** **Exception** **as** e**:**  **print(**f"Ошибка загрузки фона: {e}"**)**  **def** \_resize\_background**(**self**,** event**=None):**  """Масштабирование фонового изображения под текущий размер окна"""  **if** self**.**bg\_image\_raw**:**  # Получаем текущие размеры окна  width **=** self**.**winfo\_width**()**  height **=** self**.**winfo\_height**()**  **if** width **>** 0 **and** height **>** 0**:**  # Масштабируем изображение с сохранением пропорций  image **=** self**.**bg\_image\_raw**.**copy**()**  image**.**thumbnail**((**width**,** height**),** Image**.**Resampling**.**LANCZOS**)**  # Создаем изображение для Tkinter  self**.**bg\_image **=** ImageTk**.**PhotoImage**(**image**)**  self**.**bg\_label**.**config**(**image**=**self**.**bg\_image**)**  self**.**bg\_label**.**lower**()** # Отправляем фон на задний план    **def** ask\_value\_in\_range**(**self**,** title**,** prompt**,** initial\_value**,** min\_value**,** max\_value**,** ask\_float**=False):**  """Диалог ввода числа в заданном диапазоне."""  **while** **True:**  **if** ask\_float**:**  value **=** simpledialog**.**askfloat**(**title**,** prompt**,** initialvalue**=**initial\_value**)**  **else:**  value **=** simpledialog**.**askinteger**(**title**,** prompt**,** initialvalue**=**initial\_value**)**  **if** value **is** **None:** # Если нажата "Отмена"  **return** **None**  **if** min\_value **<=** value **<=** max\_value **or** value **==** 0**:**  **return** value  tk**.**messagebox**.**showerror**(**"Ошибка"**,** f"Допустимый диапазон: {min\_value}-{max\_value}"**)**  **def** on\_entry\_click**(**self**,** event**,** regs**,** min\_val**=**13**,** max\_val**=**100**,** entry**=None,** ask\_float**=False):**  """Обработчик клика по Entry с настраиваемым диапазоном."""  current\_value **=** entry**.**get**()**  **try:**  current\_value **=** current\_value  **except** **ValueError:**  current\_value **=** min\_val # Значение по умолчанию  new\_value **=** self**.**ask\_value\_in\_range**(**  title**=**"Ввод частоты"**,**  prompt**=**f"Введите значение ({min\_val}-{max\_val}):"**,**  initial\_value**=**current\_value**,**  min\_value**=**min\_val**,**  max\_value**=**max\_val**,**  ask\_float**=**ask\_float  **)**  **if** new\_value **is** **not** **None:**  **if** ask\_float**:**  f\_bytes **=** struct**.**pack**(**'<f'**,** new\_value**)** # Упаковка float в байты  f\_uint **=** struct**.**unpack**(**'<I'**,** f\_bytes**)[**0**]** # Преобразование в unsigned int  f\_hex **=** f"{f\_uint:08x}" # Форматирование в 8-значный hex (с ведущими нулями)  # Теперь можно безопасно разбивать:  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**regs**]** **=** **int(**f\_hex**[**4**:**8**],** 16**)** # Младшие 2 байта  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**regs **+** 1**]** **=** **int(**f\_hex**[**0**:**4**],** 16**)** # Старшие 2 байта  **else:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**regs**]** **=** new\_value  **def** get\_float\_from\_registers**(**self**,** start\_reg**):**  """Получение float значения из двух 16-битных регистров"""  **try:**  register0 **=** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_reg**]**  register1 **=** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_reg **+** 1**]**  uint32 **=** **(**register1 **<<** 16**)** **|** register0  **return** struct**.**unpack**(**'<f'**,** struct**.**pack**(**'<I'**,** uint32**))[**0**]**  **except** **(KeyError,** **IndexError,** struct**.**error**)** **as** e**:**  **print(**f"Error reading registers {start\_reg}-{start\_reg**+**1}: {e}"**)**  **return** 0.0 # Значение по умолчанию при ошибке  **def** start\_mode\_func**(**self**,** button**,** reg**):**  **if** button**.**cget**(**"style"**)** **==** "NonActive.TButton"**:**  **if** reg **in** **[**START\_AUTOMAT\_N3\_CYCLE\_REG**,** START\_AUTOMAT\_N3\_MANUAL\_REG**,** START\_AUTOMAT\_N3\_STAT\_REG**]:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_CYCLE\_REG**]** **=** 1  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_MANUAL\_REG**]** **=** 1  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_STAT\_REG**]** **=** 1  **else:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**reg**]** **=** 1  button**.**configure**(**style**=**"Active.TButton"**)**  **else:**  **if** reg **in** **[**START\_AUTOMAT\_N3\_CYCLE\_REG**,** START\_AUTOMAT\_N3\_MANUAL\_REG**,** START\_AUTOMAT\_N3\_STAT\_REG**]:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_CYCLE\_REG**]** **=** 0  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_MANUAL\_REG**]** **=** 0  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**START\_AUTOMAT\_N3\_STAT\_REG**]** **=** 0  **else:**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**reg**]** **=** 0  button**.**configure**(**style**=**"NonActive.TButton"**)**  **def** update\_button\_state\_by\_register**(**self**,** start\_n3\_reg**,** start\_mode\_reg**):**  **if** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_n3\_reg**]** **==** 1**:**  self**.**start\_automat\_n3**.**configure**(**style**=**"Active.TButton"**)**  **if** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_n3\_reg**]** **==** 0**:**  self**.**start\_automat\_n3**.**configure**(**style**=**"NonActive.TButton"**)**  **if** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_mode\_reg**]** **==** 1**:**  self**.**start\_mode**.**configure**(**style**=**"Active.TButton"**)**  **if** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**start\_mode\_reg**]** **==** 0**:**  self**.**start\_mode**.**configure**(**style**=**"NonActive.TButton"**)**  **def** refresh\_entry**(**self**,** entry**,** reg**,** is\_float**):**  **if** is\_float**:**  new\_value **=** self**.**get\_float\_from\_registers**(**reg**)**  **else:**  new\_value **=** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**reg**]**  entry**.**config**(**state**=**"normal"**)**  entry**.**delete**(**0**,** tk**.**END**)**  entry**.**insert**(**0**,** **str(**new\_value**))**  entry**.**config**(**state**=**"readonly"**)**  **def** update\_back\_button\_state**(**self**,** btn**):**  # Получаем значение из хранилища  work\_value **=** self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**WORK**]**  # Устанавливаем состояние кнопки  **if** work\_value **==** 0**:**  btn**.**config**(**state**=**"disabled"**)**  **else:**  btn**.**config**(**state**=**"normal"**)** |

Листинг П3.5. Файл CycleMode.py

|  |
| --- |
| **from** frames**.**BaseFrame **import** BaseFrame  **from** tkinter **import** ttk  **from** utils**.**PressureGraph **import** PressureGraph  **from** utils**.**constants\_for\_regs **import** **\***  **class** **CycleMode(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  self**.**set\_background**(**'imgs/cycle\_mode.png'**)**  # label = ttk.Label(self, text="Циклический режим", font=('Helvetica', 14))  # label.pack(pady=20)  # Кнопка возврата к настройкам  self**.**btn\_settings **=** ttk**.**Button**(**self**,** text**=**"Настройки\n режима"**,**  command**=lambda:** controller**.**show\_frame**(**"CycleSettings"**),** style**=**"Second.TButton"**)**  self**.**btn\_settings**.**place**(**x**=**10**,** y**=**10**,** width**=**120**,** height**=**50**)**  self**.**mn1\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_mpa\_var**)**  self**.**mn1\_mpa**.**place**(**x**=**128**,** y**=**96**,** width**=**76**)**  self**.**mn1\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_kgs\_var**)**  self**.**mn1\_kgs**.**place**(**x**=**319**,** y**=**97**,** width**=**84**)**  self**.**mn2\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_mpa\_var**)**  self**.**mn2\_mpa**.**place**(**x**=**128**,** y**=**176**,** width**=**76**)**  self**.**mn2\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_kgs\_var**)**  self**.**mn2\_kgs**.**place**(**x**=**319**,** y**=**177**,** width**=**84**)**  self**.**current\_speed **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,**  textvariable**=**self**.**controller**.**speed\_mpa\_var**)**  self**.**current\_speed**.**place**(**x**=**655**,** y**=**177**,** width**=**68**)**  self**.**start\_automat\_n3 **=** ttk**.**Button**(**self**,** text**=**"Запуск автомат Н3"**,** style**=**"NonActive.TButton"**,**  command**=lambda:**  self**.**start\_mode\_func**(**self**.**start\_automat\_n3**,** START\_AUTOMAT\_N3\_CYCLE\_REG**))**  self**.**start\_automat\_n3**.**place**(**x**=**630**,** y**=**260**,** width**=**120**,** height**=**50**)**  self**.**start\_mode **=** ttk**.**Button**(**self**,** text**=**"Запуск режима"**,**  command**=lambda:** self**.**start\_mode\_func**(**self**.**start\_mode**,** START\_MODE\_CYCLE\_REG**),**  style**=**"NonActive.TButton"**)**  self**.**start\_mode**.**place**(**x**=**630**,** y**=**360**,** width**=**120**,** height**=**50**)**  self**.**cycles **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0"**,** textvariable**=**self**.**controller**.**number\_of\_cycles\_var**)**  self**.**cycles**.**place**(**x**=**655**,** y**=**97**,** width**=**68**)**  # Инициализация графика  self**.**pressure\_graph **=** PressureGraph**(**self**)**  **def** update\_widgets**(**self**):**  mn1\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN1**)**  self**.**controller**.**mn1\_mpa\_var**.set(round(**mn1\_value**,** 1**))**  # Обновление MN2  mn2\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN2**)**  self**.**controller**.**mn2\_mpa\_var**.set(round(**mn2\_value**,** 1**))**  # Обновление скорости  speed\_value **=** self**.**get\_float\_from\_registers**(**SPEED**)**  self**.**controller**.**speed\_mpa\_var**.set(round(**speed\_value**,** 1**))**  # Обновление циклов  self**.**controller**.**number\_of\_cycles\_var**.set(**self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**NUMBER\_OF\_CYCLES**])**  self**.**update\_back\_button\_state**(**self**.**btn\_settings**)** |

Листинг П3.6. Файл CycleSettings.py

|  |
| --- |
| **from** frames**.**BaseFrame **import** BaseFrame  **from** tkinter **import** ttk  **from** utils**.**constants\_for\_regs **import** **\***  **class** **CycleSettings(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  self**.**set\_background**(**'imgs/cycle\_settings.png'**)**  # Здесь можно добавить настройки для циклического режима  self**.**ent\_pressure\_end **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_pressure\_end**.**insert**(**0**,** "0.0"**)**  self**.**ent\_pressure\_end**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** PRESSURE\_END\_CYCLE**,** min\_val**=**PRESSURE\_END\_MIN\_MAX**[**0**],**  max\_val**=**PRESSURE\_END\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_pressure\_end**,**  ask\_float**=True))**  self**.**ent\_pressure\_end**.**place**(**x**=**454**,** y**=**94**,** width**=**112**)**  self**.**ent\_pressure\_speed **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_pressure\_speed**.**insert**(**0**,** "0.0"**)**  self**.**ent\_pressure\_speed**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** PRESSURE\_SPEED\_CYCLE**,** min\_val**=**PRESSURE\_SPEED\_MIN\_MAX**[**0**],**  max\_val**=**PRESSURE\_SPEED\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_pressure\_speed**,**  ask\_float**=True))**  self**.**ent\_pressure\_speed**.**place**(**x**=**454**,** y**=**174**,** width**=**112**)**  self**.**ent\_time\_pause **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_time\_pause**.**insert**(**0**,** "0"**)**  self**.**ent\_time\_pause**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** TIME\_PAUSE\_CYCLE **,** min\_val**=**TIME\_WAIT\_MIN\_MAX**[**0**],**  max\_val**=**TIME\_WAIT\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_time\_pause**,**  ask\_float**=False))**  self**.**ent\_time\_pause**.**place**(**x**=**454**,** y**=**254**,** width**=**112**)**  self**.**cycle\_need **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**cycle\_need**.**insert**(**0**,** "0"**)**  self**.**cycle\_need**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** CYCLES\_NEED\_CYCLE **,** min\_val**=**TIME\_WAIT\_MIN\_MAX**[**0**],**  max\_val**=**TIME\_WAIT\_MIN\_MAX**[**1**],**  entry**=**self**.**cycle\_need**,**  ask\_float**=False))**  self**.**cycle\_need**.**place**(**x**=**454**,** y**=**330**,** width**=**112**)**  self**.**reset\_cycle\_set **=** ttk**.**Button**(**self**,** text**=**"Сбросить"**,** command**=**self**.**reset\_cycle\_set\_func**,**  style**=**"NonActive.TButton"**)**  self**.**reset\_cycle\_set**.**place**(**x**=**450**,** y**=**406**,** width**=**120**,** height**=**50**)**  # Кнопки  btn\_back **=** ttk**.**Button**(**self**,** text**=**"Выбор режима"**,**  command**=lambda:** controller**.**show\_frame**(**"MainMenu"**),** style**=**"Second.TButton"**)**  btn\_start **=** ttk**.**Button**(**self**,** text**=**"Запуск режима"**,**  command**=lambda:** controller**.**show\_frame**(**"CycleMode"**),** style**=**"Second.TButton"**)**  btn\_back**.**place**(**x**=**10**,** y**=**10**,** width**=**120**,** height**=**50**)**  btn\_start**.**place**(**x**=**601**,** y**=**12**,** width**=**120**,** height**=**50**)**  **def** reset\_cycle\_set\_func**(**self**):**  self**.**controller**.**slave**.**data\_store**[**"holding\_registers"**][**DROP\_NUMBER\_OF\_CYCLES**]** **=** 1  **def** update\_widgets**(**self**):**  self**.**refresh\_entry**(**self**.**ent\_pressure\_end**,** PRESSURE\_END\_CYCLE**,** is\_float**=True)**  self**.**refresh\_entry**(**self**.**ent\_pressure\_speed**,** PRESSURE\_SPEED\_CYCLE**,** is\_float**=True)**  self**.**refresh\_entry**(**self**.**ent\_time\_pause**,** TIME\_PAUSE\_CYCLE**,** is\_float**=False)**  self**.**refresh\_entry**(**self**.**cycle\_need**,** CYCLES\_NEED\_CYCLE**,** is\_float**=False)** |

Листинг П3.7. Файл MainMenu.py

|  |
| --- |
| **from** frames**.**BaseFrame **import** BaseFrame  **from** tkinter **import** ttk  **from** utils**.**constants\_for\_regs **import** **\***  **class** **MainMenu(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  self**.**set\_background**(**"imgs/main\_menu.png"**)**  # Кнопки главного меню  btn\_stat **=** ttk**.**Button**(**self**,** text**=**"Статический режим"**,**  command**=lambda:** controller**.**show\_frame**(**"StatSettings"**),** style**=**"Main.TButton"**)**  btn\_cycle **=** ttk**.**Button**(**self**,** text**=**"Циклический режим"**,**  command**=lambda:** controller**.**show\_frame**(**"CycleSettings"**),** style**=**"Main.TButton"**)**  btn\_manual **=** ttk**.**Button**(**self**,** text**=**"Ручной режим"**,**  command**=lambda:** controller**.**show\_frame**(**"ManualMode"**),** style**=**"Main.TButton"**)**  btn\_stat**.**place**(**x**=**200**,** y**=**130**,** width**=**395**,** height**=**68**)**  btn\_cycle**.**place**(**x**=**200**,** y**=**240**,** width**=**395**,** height**=**68**)**  btn\_manual**.**place**(**x**=**200**,** y**=**350**,** width**=**395**,** height**=**68**)**  **def** update\_widgets**(**self**):**  **pass** |

Листинг П3.8. Файл ManualMode.py

|  |
| --- |
| **import** struct  **import** tkinter  **from** tkinter **import** ttk  **from** frames**.**BaseFrame **import** BaseFrame  **from** utils**.**PressureGraph **import** PressureGraph  **from** utils**.**constants\_for\_regs **import** **\***  **class** **ManualMode(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  self**.**set\_background**(**'imgs/manual\_mode.png'**)**  self**.**test\_counter **=** 0  # Основные элементы интерфейса  # Кнопка возврата  self**.**btn\_settings **=** ttk**.**Button**(**self**,** text**=**"Выбор режима"**,**  command**=lambda:** self**.**controller**.**show\_frame**(**"MainMenu"**),**  style**=**"Second.TButton"**)**  self**.**btn\_settings**.**place**(**x**=**10**,** y**=**10**,** width**=**120**,** height**=**50**)**  # Метки и поля ввода  self**.**mn1\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_mpa\_var**)**  self**.**mn1\_mpa**.**place**(**x**=**128**,** y**=**96**,** width**=**76**)**  self**.**mn1\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_kgs\_var**)**  self**.**mn1\_kgs**.**place**(**x**=**319**,** y**=**97**,** width**=**84**)**  self**.**mn2\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_mpa\_var**)**  self**.**mn2\_mpa**.**place**(**x**=**128**,** y**=**176**,** width**=**76**)**  self**.**mn2\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_kgs\_var**)**  self**.**mn2\_kgs**.**place**(**x**=**319**,** y**=**177**,** width**=**84**)**  self**.**current\_speed **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**speed\_mpa\_var**)**  self**.**current\_speed**.**place**(**x**=**655**,** y**=**177**,** width**=**66**)**  # Кнопки управления  self**.**start\_automat\_n3 **=** ttk**.**Button**(**self**,** text**=**"Запуск автомат Н3"**,**  style**=**"NonActive.TButton"**,**  command**=lambda** **:**self**.**start\_mode\_func**(**self**.**start\_automat\_n3**,** START\_AUTOMAT\_N3\_MANUAL\_REG**))**  self**.**start\_automat\_n3**.**place**(**x**=**630**,** y**=**260**,** width**=**120**,** height**=**50**)**  self**.**start\_mode **=** ttk**.**Button**(**self**,** text**=**"Запуск режима"**,**  command**=lambda:** self**.**start\_mode\_func**(**self**.**start\_mode**,** START\_MODE\_MANUAL\_REG**),**  style**=**"NonActive.TButton"**)**  self**.**start\_mode**.**place**(**x**=**630**,** y**=**360**,** width**=**120**,** height**=**50**)**  self**.**ent\_frequency\_percent **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,**  font**=(**'Times New Roman'**,** 22**),**  justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_frequency\_percent**.**insert**(**0**,** "0"**)**  self**.**ent\_frequency\_percent**.**place**(**x**=**654**,** y**=**94**,** width**=**72**,** height**=**42**)**  self**.**ent\_frequency\_percent**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** FREQ\_MANUAL**,** min\_val**=**FREQ\_MANUAL\_MIN\_MAX**[**0**],**  max\_val**=**FREQ\_MANUAL\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_frequency\_percent**,**  ask\_float**=False))**  # Инициализация графика  self**.**pressure\_graph **=** PressureGraph**(**self**)**  **def** update\_widgets**(**self**):**  mn1\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN1**)**  self**.**controller**.**mn1\_mpa\_var**.set(round(**mn1\_value**,** 1**))**  # Обновление MN2  mn2\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN2**)**  self**.**controller**.**mn2\_mpa\_var**.set(round(**mn2\_value**,** 1**))**  # Обновление скорости  speed\_value **=** self**.**get\_float\_from\_registers**(**SPEED**)**  self**.**controller**.**speed\_mpa\_var**.set(round(**speed\_value**,** 1**))**  self**.**refresh\_entry**(**self**.**ent\_frequency\_percent**,** FREQ\_MANUAL**,** is\_float**=False)**  self**.**update\_back\_button\_state**(**self**.**btn\_settings**)** |

Листинг П3.9. Файл StatMode.py

|  |
| --- |
| **from** frames**.**BaseFrame **import** BaseFrame  **from** tkinter **import** ttk  **from** utils**.**PressureGraph **import** PressureGraph  **from** utils**.**constants\_for\_regs **import** **\***  **class** **StatMode(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  self**.**set\_background**(**'imgs/stat\_mode.png'**)**  # Кнопка возврата к настройкам  self**.**btn\_settings **=** ttk**.**Button**(**self**,** text**=**"Настройки\n режима"**,**  command**=lambda:** controller**.**show\_frame**(**"StatSettings"**),** style**=**"Second.TButton"**)**  self**.**btn\_settings**.**place**(**x**=**10**,** y**=**10**,** width**=**120**,** height**=**50**)**    self**.**mn1\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_mpa\_var**)**  self**.**mn1\_mpa**.**place**(**x**=**128**,** y**=**96**,** width**=**76**)**  self**.**mn1\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn1\_kgs\_var**)**  self**.**mn1\_kgs**.**place**(**x**=**319**,** y**=**97**,** width**=**84**)**  self**.**mn2\_mpa **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_mpa\_var**)**  self**.**mn2\_mpa**.**place**(**x**=**128**,** y**=**176**,** width**=**76**)**  self**.**mn2\_kgs **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**mn2\_kgs\_var**)**  self**.**mn2\_kgs**.**place**(**x**=**319**,** y**=**177**,** width**=**84**)**  self**.**current\_speed **=** ttk**.**Label**(**self**,** style**=**"Main.TLabel"**,** text**=**"0.0"**,** textvariable**=**self**.**controller**.**speed\_mpa\_var**)**  self**.**current\_speed**.**place**(**x**=**655**,** y**=**177**,** width**=**66**)**  self**.**start\_automat\_n3 **=** ttk**.**Button**(**self**,** text**=**"Запуск автомат Н3"**,** style**=**"NonActive.TButton"**,**  command**=lambda** **:**self**.**start\_mode\_func**(**self**.**start\_automat\_n3**,** START\_AUTOMAT\_N3\_STAT\_REG**))**  self**.**start\_automat\_n3**.**place**(**x**=**630**,** y**=**260**,** width**=**120**,** height**=**50**)**  self**.**start\_mode **=** ttk**.**Button**(**self**,** text**=**"Запуск режима"**,** command**=lambda:** self**.**start\_mode\_func**(**self**.**start\_mode**,** START\_MODE\_STAT\_REG**),**  style**=**"NonActive.TButton"**)**  self**.**start\_mode**.**place**(**x**=**630**,** y**=**360**,** width**=**120**,** height**=**50**)**  # Инициализация графика  self**.**pressure\_graph **=** PressureGraph**(**self**)**  **def** update\_widgets**(**self**):**  mn1\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN1**)**  self**.**controller**.**mn1\_mpa\_var**.set(round(**mn1\_value**,** 1**))**  # Обновление MN2  mn2\_value **=** self**.**get\_float\_from\_registers**(**PRESSURE\_MN2**)**  self**.**controller**.**mn2\_mpa\_var**.set(round(**mn2\_value**,** 1**))**  # Обновление скорости  speed\_value **=** self**.**get\_float\_from\_registers**(**SPEED**)**  self**.**controller**.**speed\_mpa\_var**.set(round(**speed\_value**,** 1**))**  self**.**update\_back\_button\_state**(**self**.**btn\_settings**)** |

Листинг П3.10. Файл StatSettings.py

|  |
| --- |
| **from** frames**.**BaseFrame **import** BaseFrame  **from** tkinter **import** ttk  **from** utils**.**constants\_for\_regs **import** **\***  **class** **StatSettings(**BaseFrame**):**  **def** \_\_init\_\_**(**self**,** parent**,** controller**):**  **super().**\_\_init\_\_**(**parent**,** controller**)**  # Здесь можно добавить настройки для статического режима  self**.**set\_background**(**"imgs/stat\_settings.png"**)**  # Кнопки  btn\_back **=** ttk**.**Button**(**self**,** text**=**"Выбор\nрежима"**,**  command**=lambda:** controller**.**show\_frame**(**"MainMenu"**),** style**=**"Second.TButton"**)**  btn\_start **=** ttk**.**Button**(**self**,** text**=**"Запуск\nрежима"**,**  command**=lambda:** controller**.**show\_frame**(**"StatMode"**),** style**=**"Second.TButton"**)**  btn\_back**.**place**(**x**=**12**,** y**=**11**,** width**=**120**,** height**=**50**)**  btn\_start**.**place**(**x**=**601**,** y**=**11**,** width**=**120**,** height**=**50**)**  self**.**ent\_pressure\_end **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_pressure\_end**.**insert**(**0**,** "0.0"**)**  self**.**ent\_pressure\_end**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** PRESSURE\_END\_STAT**,** min\_val**=**PRESSURE\_END\_MIN\_MAX**[**0**],**  max\_val**=**PRESSURE\_END\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_pressure\_end**,**  ask\_float**=True))**  self**.**ent\_pressure\_end**.**place**(**x**=**454**,** y**=**94**,** width**=**112**)**  self**.**ent\_pressure\_mid **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_pressure\_mid**.**insert**(**0**,** "0.0"**)**  self**.**ent\_pressure\_mid**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** PRESSURE\_MID\_STAT**,** min\_val**=**PRESSURE\_END\_MIN\_MAX**[**0**],**  max\_val**=**PRESSURE\_END\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_pressure\_mid**,**  ask\_float**=True))**  self**.**ent\_pressure\_mid**.**place**(**x**=**454**,** y**=**174**,** width**=**112**)**  self**.**ent\_speed **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**ent\_speed**.**insert**(**0**,** "0.0"**)**  self**.**ent\_speed**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** PRESSURE\_SPEED\_STAT**,** min\_val**=**PRESSURE\_SPEED\_MIN\_MAX**[**0**],**  max\_val**=**PRESSURE\_SPEED\_MIN\_MAX**[**1**],**  entry**=**self**.**ent\_speed**,**  ask\_float**=True))**  self**.**ent\_speed**.**place**(**x**=**454**,** y**=**254**,** width**=**112**)**  self**.**time\_wait\_1 **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**time\_wait\_1**.**insert**(**0**,** "0"**)**  self**.**time\_wait\_1**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** TIME\_WAIT\_1\_STAT **,** min\_val**=**TIME\_WAIT\_MIN\_MAX**[**0**],**  max\_val**=**TIME\_WAIT\_MIN\_MAX**[**1**],**  entry**=**self**.**time\_wait\_1**,**  ask\_float**=False))**  self**.**time\_wait\_1**.**place**(**x**=**454**,** y**=**334**,** width**=**112**)**  self**.**time\_wait\_2 **=** ttk**.**Entry**(**self**,** style**=**"Main.TEntry"**,** font**=(**'Times New Roman'**,** 24**),** justify**=**"center"**,**  state**=**"readonly"**)**  self**.**time\_wait\_2**.**insert**(**0**,** "0"**)**  self**.**time\_wait\_2**.**bind**(**"<Button-1>"**,**  **lambda** e**:** self**.**on\_entry\_click**(**e**,** TIME\_WAIT\_2\_STAT**,** min\_val**=**TIME\_WAIT\_MIN\_MAX**[**0**],**  max\_val**=**TIME\_WAIT\_MIN\_MAX**[**1**],**  entry**=**self**.**time\_wait\_2**,**  ask\_float**=False))**  self**.**time\_wait\_2**.**place**(**x**=**454**,** y**=**414**,** width**=**112**)**  **def** update\_widgets**(**self**):**  self**.**refresh\_entry**(**self**.**ent\_pressure\_end**,** PRESSURE\_END\_STAT**,** is\_float**=True)**  self**.**refresh\_entry**(**self**.**ent\_pressure\_mid**,** PRESSURE\_MID\_STAT**,** is\_float**=True)**  self**.**refresh\_entry**(**self**.**ent\_speed**,** PRESSURE\_SPEED\_STAT**,** is\_float**=True)**  self**.**refresh\_entry**(**self**.**time\_wait\_1**,** TIME\_WAIT\_1\_STAT**,** is\_float**=False)**  self**.**refresh\_entry**(**self**.**time\_wait\_2**,** TIME\_WAIT\_2\_STAT**,** is\_float**=False)** |

# Приложение 4

**ТАБЛИЦА ЗАПРОСОВ И ОТВЕТОВ В MODBUS**

Таблица П4.1

Команда на чтение аналогового входа 0x03

|  |  |
| --- | --- |
| **Запрос от ведущего** | **Ответ от ведомого** |
| Адрес устройства | Адрес устройства |
| Функциональный код (0x03) | Функциональный код (x03) |
| Адрес первого регистра Hi байт | Количество байт далее |
| Адрес первого регистра Lo байт | Значение регистра Hi |
| Количество регистров Hi байт | Значение регистра Lo |
| Количество регистров Lo байт | … |
| Контрольная сумма CRC | … |
| Контрольная сумма CRC | Значение регистра Hi |
|  | Значение регистра Lo |
| Контрольная сумма CRC |
| Контрольная сумма CRC |

Таблица П4.2

Команда на запись аналогового входа 0x06

|  |  |
| --- | --- |
| **Запрос от ведущего** | **Ответ от ведомого** |
| Адрес устройства | Адрес устройства |
| Функциональный код (0x06) | Функциональный код (x06) |
| Адрес первого регистра Hi байт | Адрес первого регистра Hi байт |
| Адрес первого регистра Lo байт | Адрес первого регистра Lo байт |
| Значение Hi байт | Значение Hi байт |
| Значение Lo байт | Значение Lo байт |
| Контрольная сумма CRC | Контрольная сумма CRC |
| Контрольная сумма CRC | Контрольная сумма CRC |

Таблица П4.3

Команда на запись нескольких аналоговых входов 0x10

|  |  |
| --- | --- |
| **Запрос от ведущего** | **Ответ от ведомого** |
| Адрес устройства | Адрес устройства |
| Функциональный код | Функциональный код (x06) |
| Адрес первого регистра Hi байт | Адрес первого регистра Hi байт |
| Адрес первого регистра Lo байт | Адрес первого регистра Lo байт |
| Количество регистров Hi байт | Количество записанных рег. Hi байт |
| Количество регистров Lo байт | Количество записанных рег. Lo байт |
| Количество байт далее | Контрольная сумма CRC |
| Значение Hi | Контрольная сумма CRC |
| Значение Lo |  |
| … |
| … |
| Значение Hi |
| Значение Lo |
| Контрольная сумма CRC |
| Контрольная сумма CRC |

# Приложение 5

**Листинг Программы ПР200**

Листинг П5.1. Функциональный блок int\_logic\_

|  |
| --- |
| function\_block int\_logic\_   var\_input //объявление входных переменных  I1 : udint; // Вход для первой переменной  I2 : udint; // Вход для второй переменной  RES : bool; // Сброс  end\_var   var\_output //объявление выходных переменных  Q : udint; //выходная переменная с типом данных udint  end\_var   var //объявление локальных переменных  I1\_mas : array [0..2] of udint;  I2\_mas : array [0..2] of udint;  end\_var    if RES then  I1\_mas[0] := 0;  I1\_mas[1] := 0;  I1\_mas[2] := 0;  I2\_mas[0] := 0;  I2\_mas[1] := 0;  I2\_mas[2] := 0;  Q := 0;  return;  end\_if    I2\_mas[2] := I2\_mas[1];  I2\_mas[1] := I2\_mas[0];  I2\_mas[0] := I2;  I1\_mas[2] := I1\_mas[1];  I1\_mas[1] := I1\_mas[0];  I1\_mas[0] := I1;    if (I2\_mas[0] <> I2\_mas[1]) and (I2\_mas[1] = I2\_mas[2]) and (I2\_mas[0] <> Q) then  Q := I2\_mas[0];  end\_if    if (I1\_mas[0] <> I1\_mas[1]) and (I1\_mas[1] = I1\_mas[2]) and (I1\_mas[0] <> Q) then  Q := I1\_mas[0];  end\_if  end\_function\_block |

Листинг П5.2. Функциональный блок small\_control\_

|  |
| --- |
| function\_block small\_control\_    var\_input //объявление входных переменных  button : bool;  cycle\_i : bool;  static\_i : bool;  manual\_i : bool;  start\_main\_pump : bool;  percent\_main\_pump : real;  end\_var   var\_output //объявление выходных переменных  small\_pump : bool;  manual\_o : bool;  cycle\_o : bool;  static\_o : bool;  end\_var   var //объявление локальных переменных  TON\_delay\_enable\_pump : SYS.TON;  TON\_delay\_disable\_pump : SYS.TON;  state : udint := 0; // 0 - prepare // 1 - enable // 2 - disable  //  cycle\_i\_change : array [0..1] of bool;  static\_i\_change : array [0..1] of bool;  manual\_i\_change : array [0..1] of bool;  //  small\_pump\_without\_main : bool := false;  small\_pump\_without\_main\_change : array [0..1] of bool;  //  small\_pump\_with\_main : bool := false;  small\_pump\_with\_main\_change : array [0..1] of bool;  small\_pump\_with\_main\_from\_mode : bool := false;  //  mode\_without\_main : array [0..3] of bool;  end\_var     TON\_delay\_enable\_pump(T:= T#10s);  TON\_delay\_disable\_pump(T := T#500ms);   //  cycle\_i\_change[1] := cycle\_i\_change[0];  cycle\_i\_change[0] := cycle\_i;  static\_i\_change[1] := static\_i\_change[0];  static\_i\_change[0] := static\_i;  manual\_i\_change[1] := manual\_i\_change[0];  manual\_i\_change[0] := manual\_i;  small\_pump\_without\_main\_change[1] := small\_pump\_without\_main\_change[0];  small\_pump\_without\_main\_change[0] := small\_pump\_without\_main;  small\_pump\_with\_main\_change[1] := small\_pump\_with\_main\_change[0];  small\_pump\_with\_main\_change[0] := small\_pump\_with\_main;  mode\_without\_main[2] := mode\_without\_main[1];  mode\_without\_main[1] := mode\_without\_main[0];  //    if not start\_main\_pump then  small\_pump\_with\_main := false;  small\_pump\_with\_main\_from\_mode := false;  state := 0;    if (button  or (cycle\_i\_change[0] <> cycle\_i\_change[1])   or (static\_i\_change[0] <> static\_i\_change[1])   or (manual\_i\_change[0] <> manual\_i\_change[1]))  and small\_pump\_without\_main\_change[0] = small\_pump\_without\_main\_change[1]   and mode\_without\_main[0]   and mode\_without\_main[1]  and mode\_without\_main[2]  then    small\_pump\_without\_main := not small\_pump\_without\_main;  end\_if  cycle\_o := small\_pump\_without\_main;  static\_o := small\_pump\_without\_main;  manual\_o := small\_pump\_without\_main;    mode\_without\_main[0] := true;  else  // Обнуление режима без основного двигателя  small\_pump\_without\_main := false;    if (button  or (cycle\_i\_change[0] <> cycle\_i\_change[1])   or (static\_i\_change[0] <> static\_i\_change[1])   or (manual\_i\_change[0] <> manual\_i\_change[1]))  and small\_pump\_with\_main\_change[0] = small\_pump\_with\_main\_change[1]   and not mode\_without\_main[0]   and not mode\_without\_main[1]  and not mode\_without\_main[2]  then    small\_pump\_with\_main := not small\_pump\_with\_main;  end\_if  cycle\_o := small\_pump\_with\_main;  static\_o := small\_pump\_with\_main;  manual\_o := small\_pump\_with\_main;    if small\_pump\_with\_main then  case state of  0:  TON\_delay\_enable\_pump.I := false;  TON\_delay\_disable\_pump.I := false;  small\_pump\_with\_main\_from\_mode := false;  state := 1;  1:  TON\_delay\_enable\_pump.I := true;  TON\_delay\_disable\_pump.I := false;  small\_pump\_with\_main\_from\_mode := true;  if TON\_delay\_enable\_pump.Q = true then  state := 2;  end\_if  2:  TON\_delay\_enable\_pump.I := false;  TON\_delay\_disable\_pump.I := true;  small\_pump\_with\_main\_from\_mode := false;  if TON\_delay\_disable\_pump.Q = true then  state := 0;  end\_if  end\_case  else  small\_pump\_with\_main\_from\_mode := false;  end\_if    mode\_without\_main[0] := false;  end\_if    small\_pump := small\_pump\_without\_main or small\_pump\_with\_main\_from\_mode;   end\_function\_block |

Листинг П5.3. Функциональный блок small\_control\_

|  |
| --- |
| function\_block ON\_OFF\_ST\_    var\_input //объявление входных переменных  I1 : bool; // Вход кнопки  Freq : udint; // Частота  clr : bool; // Сброс состояния  end\_var   var\_output //объявление выходных переменных  Set : bool; // Установить частоту  Freq\_out : udint; // Выходная частота  end\_var   var //объявление локальных переменных  state : udint;  TON\_1\_sec: SYS.TON;  TON\_1\_sec\_event : bool;  end\_var    TON\_1\_sec(I:= TON\_1\_sec\_event, T:= t#0d0h0m1s000ms);    if clr and state <> 4 then  TON\_1\_sec\_event := false;  state := 0;  Set := false;  Freq\_out := 0;  end\_if    case state of  0:  Set := false;  if I1 = true then // Нажата кнопка  Set := true;  Freq\_out := Freq;  state := 1;  TON\_1\_sec\_event := true;  end\_if  1: // Пользователь держет  if TON\_1\_sec.Q then  state := 2;  TON\_1\_sec\_event := false;  end\_if  if I1 = false then // Отжал  Set := false;  state := 3;  TON\_1\_sec\_event := false;  end\_if  2: // Пользователь удерживает  if I1 = false then // Пользователь отжал  Freq\_out := 0;  state := 0;  end\_if  3: // Ждем второе нажатие  if I1 = true then  Set := true;  Freq\_out := 0;  state := 4;  end\_if  4: // Ждем пока пользователь отпустит кнопку  Set := false;  if I1 = false then  state := 0;  end\_if  else   state := 0;  end\_case  end\_function\_block |

Листинг П5.4. Функциональный блок button\_logic\_

|  |
| --- |
| function\_block button\_logic\_   var\_input //объявление входных переменных  I1 : bool; // Вход кнопки нажимной   I2 : bool; // Вход кнопки зажимной  I3 : bool; // Вход кнопки зажимной  RES : bool; // Сброс  end\_var   var\_output //объявление выходных переменных  Q : bool; //выходная переменная с типом данных udint  end\_var   var //объявление локальных переменных  I2\_mas : array [0..2] of bool;  I3\_mas : array [0..2] of bool;  end\_var    if RES then  I2\_mas[0] := false;  I2\_mas[1] := false;  I2\_mas[2] := false;  I3\_mas[0] := false;  I3\_mas[1] := false;  I3\_mas[2] := false;  Q := false;  return;  end\_if    I2\_mas[2] := I2\_mas[1];  I2\_mas[1] := I2\_mas[0];  I2\_mas[0] := I2;  I3\_mas[2] := I3\_mas[1];  I3\_mas[1] := I3\_mas[0];  I3\_mas[0] := I3;    if I1 then  Q := not Q;  end\_if    if (I2\_mas[0] <> I2\_mas[1]) and (I2\_mas[1] = I2\_mas[2]) and (I2\_mas[0] <> Q) then  Q := not Q;  end\_if    if (I3\_mas[0] <> I3\_mas[1]) and (I3\_mas[1] = I3\_mas[2]) and (I3\_mas[0] <> Q) then  Q := not Q;  end\_if  end\_function\_block |

Листинг П5.5. Функциональный блок calc\_speed\_

|  |
| --- |
| function\_block calc\_speed\_   var\_input //объявление входных переменных  I : real;  end\_var   var\_output // объявляение входных переменных  Q : real;  end\_var   var //объявление локальных переменных  TON\_delay : SYS.TON;  data : array [0..8] of real;  coeff : array [0..8] OF real;  first\_scan : bool := true;  j : udint;  end\_var    // Создание коэффициентов  if first\_scan then  for j := 0 to 8 do  coeff[j] := (udint\_to\_real(j) - 4) / 12;  end\_for  first\_scan := false;  end\_if    TON\_delay(T := udint\_to\_time(200));  TON\_delay.I := true;    if not TON\_delay.Q then // Таймер 200 ms  return;  end\_if    // Сдвиг значений   TON\_delay.I := false;  for j := 0 to 7 do  data[8 - j] := data[8 - j - 1];  end\_for  data[0] := I;    // Подсчет ответа  Q := 0;  for j := 0 to 8 do  Q := Q + data[j] \* coeff[j];  end\_for  end\_function\_block |

Листинг П5.6. Функциональный блок control\_cyclical\_

|  |
| --- |
| function\_block control\_cyclical\_   var\_input  start : bool; // старт алгоритма  P : real; // текущее давление  valve\_in : real; // текущее состояние клапана  P\_max : real; // итоговое давление  P\_min : real; // до какого давления сбрасывать  t\_waiting : udint; // время удержания (мин)  v : real; // текущая скорость (МПа/мин)  v\_filling : real; // скорсоть набора воды (МПа / мин)  count\_cycles : udint; // количество требуемых циклов  clear\_counter : udint; // сброс счетчика итераций  end\_var   var\_output //объявление выходных переменных  enable\_control : bool; // включить управление контролем скорости  target : real;  current : real;  counter : udint;  valve\_out : real; // на сколько открыть клапан  ended : bool;  end\_var   var //объявление локальных переменных  state : udint; // текущее состояние  // промежуточные данные  // Таймер  TON\_delay\_end\_time : SYS.TON;  end\_var    TON\_delay\_end\_time(T:= udint\_to\_time(t\_waiting \* 60 \* 1000));    if udint\_to\_bool(clear\_counter) then  counter := 0;  end\_if    if not start then  state := 0;  target := 0;  ended := false;  enable\_control := false;  TON\_delay\_end\_time.I := false;  end\_if    case state of  0: // Ожидание старта  ended := false;  enable\_control := false;  TON\_delay\_end\_time.I := false;    if start then  state := 1;  end\_if  1: // Набор  enable\_control := true;  current := v;  target := v\_filling;    if P >= P\_max then  enable\_control := false;  state := 2;  end\_if  2: // Удержание  TON\_delay\_end\_time.I := true;    if TON\_delay\_end\_time.Q then  TON\_delay\_end\_time.I := false;  counter := counter + 1;  if counter < count\_cycles then  state := 3;  valve\_out := 2.5;  else  state := 0;  ended := true;  end\_if  end\_if  3: // Сброс  if P <= P\_min then  valve\_out := 0;  state := 4;  end\_if  4: // Закрытие клапанов  if valve\_in < 0.5 then  state := 1;  end\_if  else  state := 0;  end\_case  end\_function\_block |

Листинг П5.7. Функциональный блок regulation\_

|  |
| --- |
| function\_block regulation\_   var\_input // входные переменные  enable : bool; // разрешение регуляции  target : real; // требуемое давление  current : real; // текущее давление  min : real; // минимальная частота пчв  max : real; // максимальная частота пчв  end\_var   var\_output  q : real; // управляющая частота пчв, %  end\_var   var // локальные переменные  TON\_delay : SYS.TON;  kp : real := 35.0; // коэффициент p-звена (инициализация шаг 0)  gamma : real := 0.15; // скорость адаптации  sigma : real := 0.035; // коэффициент утечки  e : real; // текущая ошибка  rel\_err : real; // относительная ошибка  range : real := 100.0; // рабочий диапазон %, для нормировки  end\_var    TON\_delay(T := udint\_to\_time(1000));    // если управление запрещено - обнуляем выход и выходим  if not enable then  q := 0.0;  TON\_delay.I := false;  return;  end\_if;  TON\_delay.I := true;  if not TON\_delay.Q then  return;  end\_if   TON\_delay.I := false;  // расчёт ошибки  e := target - current;   // σ-адаптация параметра kp: kp[k+1] = kp[k] + γ·e[k] − σ·kp[k]  kp := kp + gamma \* e - sigma \* kp;  if kp < 0.0 then kp := 0.0; end\_if; // защита от отрицательных значений   // выход регулятора: линейное «первое приближение» + корректировка p-звена  q := 0.02865982 \* target + kp \* e;   // насыщение в заданных пределах  if q > max then  q := max;  elsif q < min then  q := min;  end\_if;  end\_function\_block |

Листинг П5.8. Функциональный блок control\_static\_

|  |
| --- |
| function\_block control\_static\_  var\_input  start : bool; // старт алгоритма  P : real; // текущее давление  P\_max : real; // итоговое давление (МПа)  P\_interim : real; // промежуточное давление (МПа)  t\_1 : udint; // время до промежуточного (мин)  t\_2 : udint; // время после промежуточного (мин)  v : real; // текущая скорость (МПа/мин)  v\_filling : real; // скорость набора воды (МПа/мин)  d\_P : real; // шаг ступени (МПа)  end\_var   var\_output //объявление выходных переменных  target : real;  current : real;  ended : bool;  end\_var   var //объявление локальных переменных  state : udint; // текущее состояние  // промежуточные данные  start\_P : real; // давление с которого начинается график  end\_P : real; // давление на котором кончается график  t\_control : udint; // время которое удерживать давление  TON\_delay\_end\_time : SYS.TON;  end\_var   if not start then  state := 0;  target := 0;  ended := false;  TON\_delay\_end\_time.I := false;  end\_if    case state of  0: // Ожидание старта  target := 0;  start\_P := 0;  TON\_delay\_end\_time.I := false;  end\_P := P - udint\_to\_real(real\_to\_udint(P \* 10) MOD real\_to\_udint(d\_P \* 10)) / 10 - 0.1;  current := 0;  ended := false;    if start then  state := 1;  end\_if  1: // Обновление данных для графика  start\_P := end\_P;  end\_P := end\_P + d\_P;  if end\_P > P\_max then  end\_P := P\_max;  end\_if    t\_control := t\_1 \* 60 \* 1000;  if end\_P >= P\_interim then  t\_control := t\_2 \* 60 \* 1000;   end\_if    state := 2;  2: // Набор  target := v\_filling;  current := v;  TON\_delay\_end\_time.I := false;    if P >= end\_P then  state := 3;  TON\_delay\_end\_time(T:= udint\_to\_time(t\_control));  TON\_delay\_end\_time.I := true;  end\_if  3: // Удержание  target := end\_P;  current := P;    if TON\_delay\_end\_time.Q then  TON\_delay\_end\_time.I := false;  // Следующий этап  if end\_P = P\_max then  state := 0;  ended := true;  else  state := 1;  end\_if  end\_if  else  state := 0;  end\_case  end\_function\_block |