|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации Мытищинский филиал федерального государственного автономного образовательного учреждения высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МФ МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Космический»

КАФЕДРА «Прикладная математика, информатика и вычислительная техника»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

Разработка программного обеспечения деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Студент К3-85Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Ю. Аленина

(Подпись, дата)

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Афанасьев

(Подпись, дата)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Чернышов

(Подпись, дата)

*2025 г.*

АННОТАЦИЯ

Расчетно-пояснительная записка состоит из 143 страниц, содержит 11 листингов, 7 иллюстраций, 1 таблицу и 7 использованных источников.

Работа включает две главы, введение и заключение.

Работа посвящена разработке программного обеспечения для деревообрабатывающего станка с числовым программным управлением.

Актуальность данной работы – растущий спрос на автоматизацию в деревообработке: современное производство стремится повысить производительность, снизить затраты и улучшить качество продукции. Автоматизация, в частности использование станков с ЧПУ, является ключевым инструментом для достижения этих целей.

В рамках данной выпускной квалификационной работы была разработана и реализована система программного управления для деревообрабатывающего станка с ЧПУ, соответствующая современным требованиям к автоматизированному производству в деревообрабатывающей промышленности.

Результаты работы будут использованы при производстве деревообрабатывающего токарно-фрезерного станка с ЧПУ.

На данный момент разработка находится на стадии тестирования и испытаний.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc201402074)

[1 Общетехническое обоснование 7](#_Toc201402075)

[1.1 Описание интегрированной среды разработки Codesys 7](#_Toc201402076)

[1.2 Описание деревообрабатывающего токарно-фрезерного станка с ЧПУ 10](#_Toc201402077)

[1.3 Цель работы 12](#_Toc201402078)

[1.4 Актуальность разработки 13](#_Toc201402079)

[1.5 Постановка задачи 14](#_Toc201402080)

[2 Решение задачи 18](#_Toc201402081)

[2.1 Разработка архитектуры программного обеспечения 18](#_Toc201402082)

[2.2 Разработка алгоритмов работы для каждого функционального модуля, обеспечивающих выполнение заданных операций 26](#_Toc201402083)

[2.3 Разработка структуры данных для хранения и обработки информации, включая данные с датчиков, параметры обработки и управляющие сигналы 32](#_Toc201402084)

[2.4 Выбор оптимальных языков программирования и библиотек CODESYS для реализации каждого модуля 35](#_Toc201402085)

[2.5 Реализация функциональных модулей программы в CODESYS 38](#_Toc201402086)

[2.6 Интеграция и отладка программного обеспечения 81](#_Toc201402087)

[2.7 Обеспечение надежности и безопасности 87](#_Toc201402088)

[ЗАКЛЮЧЕНИЕ 92](#_Toc201402089)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 94](#_Toc201402090)

[ПРИЛОЖЕНИЕ А 95](#_Toc201402091)

[ПРИЛОЖЕНИЕ Б 104](#_Toc201402092)

# СПИСОК СОКРАЩЕНИЙ

АСУ – автоматизированная система управления;

ГОСТ – государственный стандарт;

М – мотор;

ПО – программное обеспечение;

ПЛК – программируемый логический контроллер;

ХВ – ходовой винт;

ШД – шаговый двигатель;

ЧПУ – числовое программное управление;

Э – энкодер.

# ВВЕДЕНИЕ

Современное производство стремится к автоматизации и повышению эффективности, особенно в таких отраслях, как деревообработка. Токарно-фрезерные станки с числовым программным управлением (ЧПУ) являются ключевым элементом для достижения этих целей, предоставляя возможность создания высокоточных изделий с высокой скоростью и минимальными потерями материала.

Разработанное программное обеспечение может быть непосредственно использовано на деревообрабатывающем станке, что подтверждает практическую ценность данной работы.

Однако, разработка программного обеспечения для управления такими станками представляет собой сложную задачу, требующую глубокого понимания как принципов работы станков ЧПУ, так и современных технологий программирования.

Данная работа посвящена разработке программного обеспечения для деревообрабатывающего токарно-фрезерного станка с ЧПУ. Цель работы - создание эффективного и удобного в использовании программного обеспечения, которое позволит оператору управлять станком, создавать и редактировать программы обработки, а также отслеживать процесс работы станка в режиме реального времени.

В работе будут рассмотрены:

* Архитектура системы управления станком с ЧПУ с описанием основных компонентов и их взаимодействия.
* Технологии программирования для разработки ПО, с акцентом на языки программирования, библиотеки и инструменты, наиболее подходящие для решения задач управления станком.
* Функциональность приложения, включая создание и редактирование программ обработки, управление параметрами обработки и отслеживание ошибок.
* Тестирование и отладка разработанного ПО, с целью обеспечения его надежности и стабильности.
* Результаты работы будут иметь практическую ценность, предоставляя оператору станка удобные и эффективные инструменты для работы с ЧПУ-станком.
* Данная работа является актуальной, поскольку способствует повышению эффективности деревообрабатывающего производства, улучшению качества продукции и расширению возможностей в области токарно-фрезерной обработки древесины.

# Общетехническое обоснование

## Описание интегрированной среды разработки Codesys

Рассмотрим интегрированную среду разработки (Integrated Development Environment, IDE) CODESYS, которая является ключевым инструментом для программирования, настройки и отладки систем автоматизации на основе программируемых логических контроллеров (ПЛК). CODESYS заслуженно занимает лидирующие позиции на рынке, предлагая широкий спектр возможностей, соответствующих международному стандарту МЭК 61131-3.

CODESYS — это универсальная и гибкая интегрированная среда разработки, предназначенная для создания приложений для автоматизации. Она поддерживает множество различных аппаратных платформ, благодаря чему разработчики могут создавать программное обеспечение для широкого спектра устройств: от небольших контроллеров до сложных промышленных систем.

Основные характеристики CODESYS:

* **Соответствие стандарту МЭК 61131-3**: CODESYS поддерживает все пять языков программирования, определенных в стандарте МЭК 61131-3:
  + IL (Instruction List) - Язык инструкций.
  + ST (Structured Text) - Структурированный текст.
  + FBD (Function Block Diagram) - Диаграмма функциональных блоков.
  + LD (лестничная диаграмма) — релейно-контактная схема (лестничная диаграмма).
  + SFC (Sequential Function Chart) - Диаграмма функциональной последовательности.
  + Это позволяет разработчикам выбирать наиболее подходящий язык в зависимости от задачи и опыта.
* **Поддержка различных аппаратных платформ**: CODESYS работает с широким спектром ПЛК от разных производителей, что обеспечивает гибкость при выборе оборудования. Она предлагает обширную библиотеку устройств (Device Repository) с описаниями и драйверами для многих популярных ПЛК.
* **Интегрированный отладчик**: встроенный отладчик позволяет выполнять пошаговую отладку кода, просматривать значения переменных, устанавливать точки останова и контролировать ход выполнения программы, что значительно упрощает процесс разработки и поиска ошибок.
* **Визуализация**: CODESYS предоставляет инструменты для создания визуализаций (HMI — человеко-машинный интерфейс), которые позволяют отображать данные, управлять оборудованием и создавать пользовательские интерфейсы.
* **Симуляция (Simulation)**: позволяет моделировать работу ПЛК и написанного кода на компьютере, не требуя физического оборудования. Это ускоряет процесс разработки и отладки.
* **Библиотеки и расширения**: поддержка широкого спектра библиотек, расширяющих функциональность среды. Это позволяет легко интегрировать сложные алгоритмы, управлять сетью, работать с различными протоколами связи и специализированными устройствами.
* **Поддержка сетевых протоколов**: CODESYS поддерживает различные сетевые протоколы, такие как Modbus TCP, EtherNet/IP, OPC UA, что позволяет легко интегрировать ПЛК в промышленные сети.

CODESYS предоставляет мощный набор инструментов для проектирования, разработки и отладки систем автоматизации. Рассмотрим основные функциональные возможности:

* **Проектирование ПЛК:**
  + Создание проекта для конкретного ПЛК, выбор оборудования из Device Repository.
  + Настройка аппаратных ресурсов (входы, выходы, сетевые интерфейсы).
  + Определение переменных и типов данных.
* **Разработка Программного Обеспечения:**
  + Написание кода на выбранном языке IEC 61131-3.
  + Использование графического языка (SFC) для структурирования кода.
  + Использование комментариев для документирования кода.
* **Конфигурирование Оборудования:**
  + Настройка сетевых параметров ПЛК.
  + Настройка связи с другими устройствами.
  + Настройка визуализаций.
* **Отладка и Тестирование:**
  + Загрузка программы на ПЛК.
  + Онлайн-мониторинг переменных в реальном времени.
  + Пошаговая отладка.
  + Просмотр состояния функциональных блоков и переменных.
* **Визуализация и HMI:**
  + Создание графических интерфейсов для отображения данных и управления оборудованием.
  + Использование различных графических элементов (кнопки, индикаторы, графики).
  + Настройка событий и реакций на пользовательские действия.
* **Управление Версиями (Version Control):**
  + Интеграция с системами управления версиями для отслеживания изменений в коде и совместной работы нескольких разработчиков.
* **Использование Библиотек:**
  + Использование библиотек, предоставляемых производителями ПЛК и другими поставщиками.

## Описание деревообрабатывающего токарно-фрезерного станка с ЧПУ

Рассмотрим описание токарно-фрезерного станка по дереву с ЧПУ, используемого для изготовления декоративных элементов, деталей мебели, элементов лестниц и т. д. Описание включает в себя основные компоненты станка, принцип его работы и технические характеристики.

Токарно-фрезерный станок по дереву с ЧПУ представляет собой автоматизированное оборудование, предназначенное для обработки деревянных заготовок с использованием как токарных, так и фрезерных операций. ЧПУ обеспечивает автоматическое управление движением инструмента и шпинделя в соответствии с заданной программой, что позволяет изготавливать сложные детали с высокой точностью и повторяемостью [11].

* Конструкция: Станок обычно состоит из следующих основных компонентов:
  + Станина (рама/основание): жесткая и устойчивая основа, обеспечивающая стабильность работы станка.
  + Шпиндель: электрический двигатель, вращающий заготовку (при токарной обработке) или инструмент (при фрезеровании).
  + Суппорт (каретка): подвижная часть станка, несущая инструмент.
  + Направляющие: обеспечивают линейное перемещение суппорта по осям (обычно X, Y и Z).
  + Приводы: электродвигатели с системами передачи для перемещения суппорта и вращения шпинделя [12].
  + Система ЧПУ: компьютерная система, управляющая работой станка в соответствии с программой обработки [11].
  + Защитные ограждения: обеспечивают безопасность оператора и предотвращают разброс стружки.
* Принцип Работы:
  + Загрузка заготовки: деревянная заготовка закрепляется в шпинделе.
  + Загрузка программы: оператор загружает программу обработки, написанную вручную. Программа содержит информацию о траектории движения инструмента, скорости вращения шпинделя, подаче, глубине резания и т. д.
  + Выполнение программы: система ЧПУ [10] (OWEN ПЛК) считывает программу, преобразованную в понятный для ПЛК формат, и управляет приводами, перемещая суппорт и вращая шпиндель в соответствии с заданными инструкциями. Управление осуществляется через цифровые и аналоговые выходы ПЛК, подключенные к приводам.
  + Обработка: инструмент (токарный резец, фрезерная фреза) удаляет материал с заготовки, формируя деталь заданной формы.
  + Смена инструмента: Система ЧПУ управляет процессом смены инструментов, используя логику, запрограммированную в CODESYS.
  + Завершение обработки: обработка завершается, когда программа обработки выполнена.

Кинематическая схема деревообрабатывающего станка с ЧПУ представлена на .

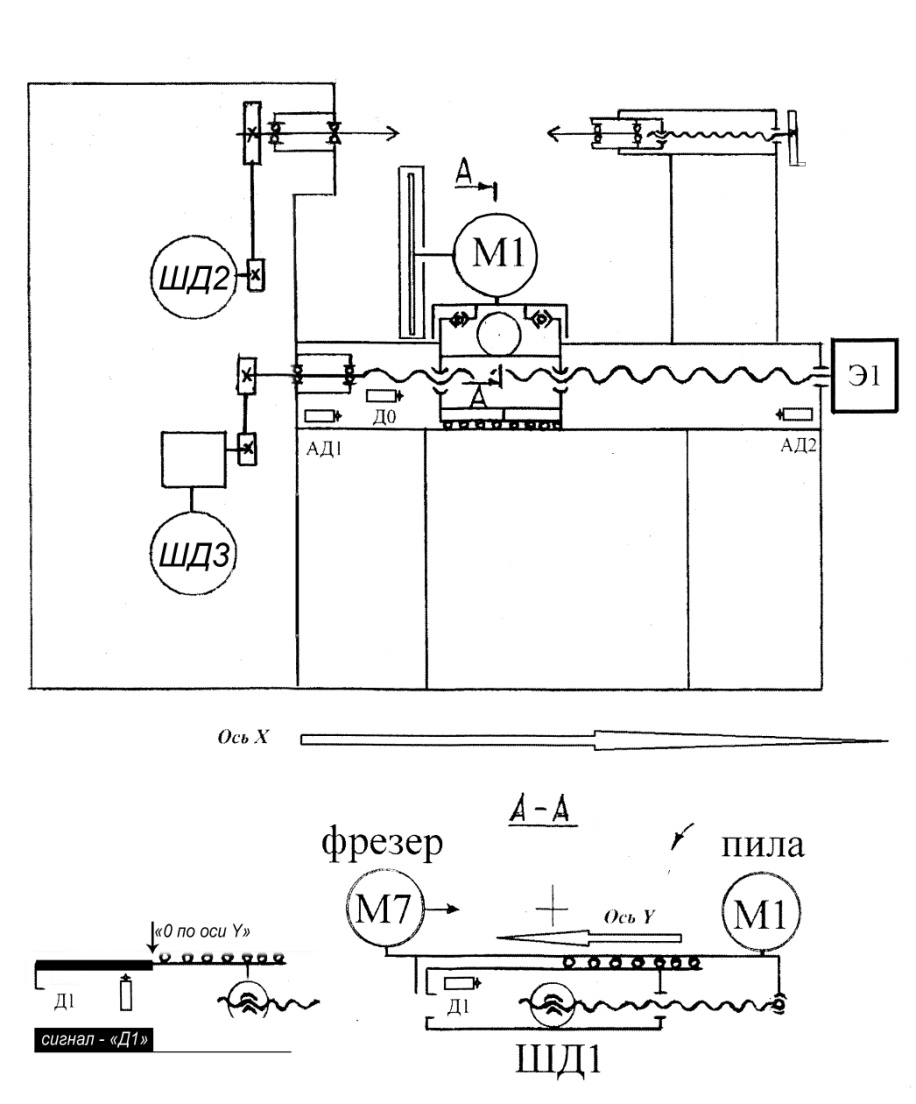


Рисунок 1 – Кинематическая схема деревообрабатывающего станка с ЧПУ.

## Цель работы

Разработка программного обеспечения для автоматизированной системы управления (АСУ) деревообрабатывающего токарно-фрезерного станка с ЧПУ, обеспечивающего управление основными исполнительными механизмами станка, контроль технологических параметров и реализацию удобного пользовательского интерфейса.

Конкретизация цели (в зависимости от вашего проекта):

* Разработка ПО для управления осями станка (X, Y).
* Разработка ПО для управления кареткой (скорость вращения).
* Разработка системы безопасности и защиты станка.
* Интеграция разработанного ПО с аппаратной частью станка (ПЛК OWEN).
* Оптимизация алгоритмов управления для повышения точности и производительности обработки.

## Актуальность разработки

Актуальность данной выпускной квалификационной работы обусловлена несколькими факторами:

* Растущий спрос на автоматизацию в деревообработке: современное производство стремится повысить производительность, снизить затраты и улучшить качество продукции. Автоматизация, в частности использование станков с ЧПУ, является ключевым инструментом для достижения этих целей [8].
* Повышение эффективности обработки материалов: деревообрабатывающие станки с ЧПУ позволяют изготавливать сложные детали с высокой точностью и повторяемостью, сокращая количество отходов и время производства.
* Необходимость в квалифицированных специалистах: развитие технологий ЧПУ требует квалифицированных специалистов, способных разрабатывать, настраивать и обслуживать системы автоматизации. Данная работа направлена на приобретение и демонстрацию таких навыков.
* Использование современных технологий: в работе будет использоваться интегрированная среда разработки CODESYS и ПЛК OWEN, что соответствует современным тенденциям в области автоматизации производства.
* Повышение конкурентоспособности предприятий: внедрение современных систем управления, разработанных в рамках данной работы, позволит предприятиям деревообрабатывающей промышленности повысить свою конкурентоспособность на рынке.
* Импортозамещение: разработка ПО на базе отечественного ПЛК (OWEN) способствует импортозамещению в сфере автоматизации производства.
* Образовательная ценность: данная работа предоставляет возможность продемонстрировать знания и навыки в области программирования, электроники, автоматизации и, в частности, программирования ПЛК, что является важной компетенцией для инженеров-автоматизаторов.
* Практическая значимость: разработанное программное обеспечение может быть непосредственно использовано на деревообрабатывающем станке, что подтверждает практическую ценность данной работы.

## Постановка задачи

В соответствии с техническим заданием на разработку программы для управления токарно-фрезерным станком КТФ-8, в рамках данной работы необходимо решить следующие задачи:

1. Разработка архитектуры программного обеспечения:

* Определение структуры программы, выделение функциональных модулей и их взаимосвязей.
* Проектирование алгоритмов работы для каждого функционального модуля, обеспечивающих выполнение заданных операций.
* Разработка структуры данных для хранения и обработки информации, включая данные с датчиков, параметры обработки и управляющие сигналы.
* Выбор оптимальных языков программирования и библиотек CODESYS для реализации каждого модуля.

2. Реализация функциональных модулей программы в CODESYS:

* Панель оператора:
  + Программирование панели оператора ИП320 для выбора режимов работы и ввода параметров.[1]
  + Разработка логики для обработки нажатий кнопок и ввода данных пользователем.
  + Реализация сохранения введенных параметров для повторного использования.
* Режимы обработки («1 проход», «2 прохода», «1 проход + фрезерование», «2 прохода + фрезерование»):
  + Разработка алгоритмов управления движением каретки (ось X) и вращением заготовки (ось Y) в соответствии с заданными режимами обработки и данными.
  + Реализация управления шаговыми двигателями ШД1, ШД2, ШД3 включая задание скорости, направления и управление переключением драйверов [5].
  + Реализация логики управления для переключения между токарной и фрезерной обработкой.
  + Реализация логики контроля датчиков (АД0, АД1) для обеспечения безопасности и правильной работы станка.
* Режим “Нулевое положение”:
  + Разработка алгоритма возврата каретки в исходное положение (координата 0) с использованием как скоростного, так и точного позиционирования.
  + Реализация управления двигателями ХВ и перемещения каретки.
  + Реализация логики контроля датчиков (Д0, Д1).
* Режим “Фрезерование”:
  + Разработка алгоритма фрезерной обработки с учётом параметров фрезерования (количество зон, количество проходов, скорость вращения заготовки, глубина погружения инструмента).
  + Реализация управления двигателем фрезера (М7).
  + Реализация управления вращением заготовки (ШД2).
  + Разработка алгоритмов расчета траектории инструмента для каждой зоны фрезерования.
* Режим “Загрузка”:
  + Разработка функции загрузки данных из текстового файла с USB-накопителя в память ПЛК.
  + Реализация обработки ошибок при загрузке данных (неверный формат, отсутствие файла и т. д.).
  + Обеспечение корректного хранения данных в памяти ПЛК.
* Режим “Ручное управление”:
  + Разработка логики управления движением каретки по осям X и Y с помощью кнопок на панели оператора ИП320.
  + Реализация отображения текущих координат на панели оператора.
  + Обеспечение безопасного перемещения каретки в ручном режиме.

3. Интеграция и отладка программного обеспечения:

* Интеграция разработанных функциональных модулей в единую систему управления.
* Отладка и тестирование программы на физическом оборудовании (станке КТФ-8) для выявления и устранения ошибок.
* Проведение испытаний для подтверждения соответствия программы требованиям технического задания.

4. Обеспечение надежности и безопасности:

* Реализация механизма защиты от некорректных действий пользователя (проверка входных данных, обработка ошибок).
* Разработка алгоритма аварийной остановки всех двигателей при нажатии кнопки «СТОП» или срабатывании аварийных датчиков.

# Решение задачи

## Разработка архитектуры программного обеспечения

На этапе разработки архитектуры ПО необходимо определить основные функциональные блоки системы и то, как они взаимодействуют друг с другом. Этот этап включает в себя выявление ключевых задач, которые должна выполнять система, и разбиение их на более мелкие, управляемые модули.

Определение функциональных требований:

Прежде всего, необходимо четко определить функциональные требования к системе.  В этом случае типичные функциональные требования могут включать:

* **Сбор данных с датчиков:** система должна собирать данные с различных датчиков.
* **Обработка данных:** система должна выполнять обработку собранных данных.
* **Принятие решений:** на основе обработанных данных система должна принимать решения о дальнейших действиях.
* **Управление устройством/процессом:** система должна выдавать управляющие сигналы для управления устройством или процессом.
* **Настройка системы:** Система должна позволять пользователю настраивать параметры работы системы, такие как скорости вращения пилы и заготовки, положение каретки, количество зон фрезерования, глубину погружения и т. д.

Выделение функциональных модулей

Исходя из определенных функциональных требований, можно выделить следующие основные функциональные модули:

* **Модуль сбора данных:** отвечает за сбор данных с датчиков.
  + Функции:
    - Инициализация датчиков.
    - Чтение данных с датчиков и кнопок панели управления.
    - Чтение данных с USB-накопителя.
    - Обработка ошибок при чтении данных.
    - Предоставление данных другим модулям.
* **Модуль обработки данных:** отвечает за обработку полученных данных.
  + Функции:
    - Фильтрация данных.
    - Преобразование данных.
    - Вычисление производных значений (скорости, ускорения).
    - Обнаружение аномалий.
* **Модуль принятия решений:** отвечает за принятие решений на основе обработанных данных.
  + Функции:
    - Анализ обработанных данных.
    - Сравнение данных с заданными порогами.
    - Определение необходимого действия.
    - Формирование управляющих сигналов.
* **Модуль управления:** отвечает за выдачу управляющих сигналов для управления устройством или процессом.
  + Функции:
    - Преобразование управляющих сигналов в формат, понятный устройству.
    - Отправка управляющих сигналов на устройство.
    - Обработка обратной связи от устройства.
* **Модуль конфигурации:** отвечает за загрузку и сохранение параметров конфигурации системы.
  + Функции:
    - Чтение параметров из файла конфигурации.
    - Сохранение параметров в файл конфигурации.
    - Предоставление интерфейса для изменения параметров.

Определение взаимосвязей между модулями:

Теперь необходимо определить, как эти модули взаимодействуют друг с другом. Диаграмма взаимосвязей может выглядеть следующим образом:

Описание взаимосвязей:

* Модуль сбора данных получает информацию с датчиков/кнопок панели и передает ее в модуль обработки данных.
* Модуль обработки данных обрабатывает полученные данные и передает их в модуль принятия решений.
* Модуль принятия решений анализирует данные и формирует управляющие сигналы, которые передаются в модуль управления.
* Модуль управления преобразует управляющие сигналы и отправляет их на устройство.
* Модуль конфигурации предоставляет параметры для всех модулей.

Функциональная схема для деревообрабатывающего токарно-фрезерного станка представлена на Рисунок 2, Рисунок 3. Инициализация: начальный этап в SFC. Он содержит процедуры инициализации процесса.

Под начальным шагом мы видим несколько блоков, выстроенных в основной последовательности. Выполнение происходит слева направо. Давайте разберём их:

Read\_Prog: Этот блок, считывает данные и значения из памяти программ в ПЛК. Он действует как входной модуль для основного рабочего процесса, в котором данные из других источников будут использоваться в основной программе.

Read\_Inputs: этот блок, считывает значения с физических входных модулей ПЛК. Эти значения будут использоваться на последующих этапах обработки.

Record\_Outputs: этот блок записывает выходное значение в файл.

Read\_bit: это считывание определенного битового значения из входного модуля.

Write\_bit: Этот блок устанавливает или сбрасывает конкретное битовое значение в ячейке памяти внутри ПЛК или в выходном модуле.

Passage\_1, Passage\_2: эти блоки, содержат основной код процесса, поэтому эти блоки представляют собой разделы программы, которые последовательно выполняются как часть основного рабочего процесса.

Использована интегрированная среда программирования Codesys для программирования ПЛК Owen.

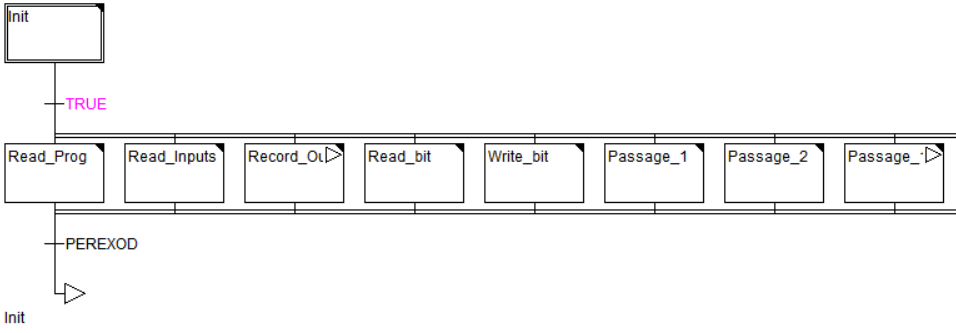


Рисунок 2 – функциональная схема станка КТФ-8.

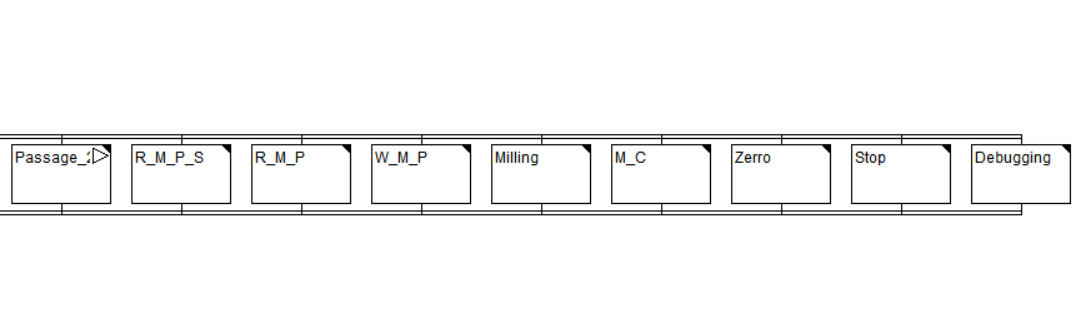


Рисунок 3 – функциональная схема станка КТФ-8.

Схема на представляет собой упрощенное отображение электрических соединений ключевых компонентов системы управления станком и демонстрирует информационный обмен между ними. Центральным элементом системы является ПЛК, который осуществляет управление всеми исполнительными механизмами и получает информацию от различных датчиков.

К ПЛК подключены следующие элементы:

* М1, М7 – мотор пилы, мотор фрезера соответственно.
* АД0, АД1 – концевые выключатели аварийной остановки всех двигателей по оси (Х).
* Д0, Д1 – датчики «0» положения продольной оси (Х), поперечной оси (Y) соответственно.
* ШД1, ШД2, ШД3 – шаговые двигатели поперечного движения каретки и заготовки и мотор ходового винта, соответственно [4].
* Э1 – энкодер на ходовой винт продольной оси (Х).
* ПАНЕЛЬ – панель оператора.

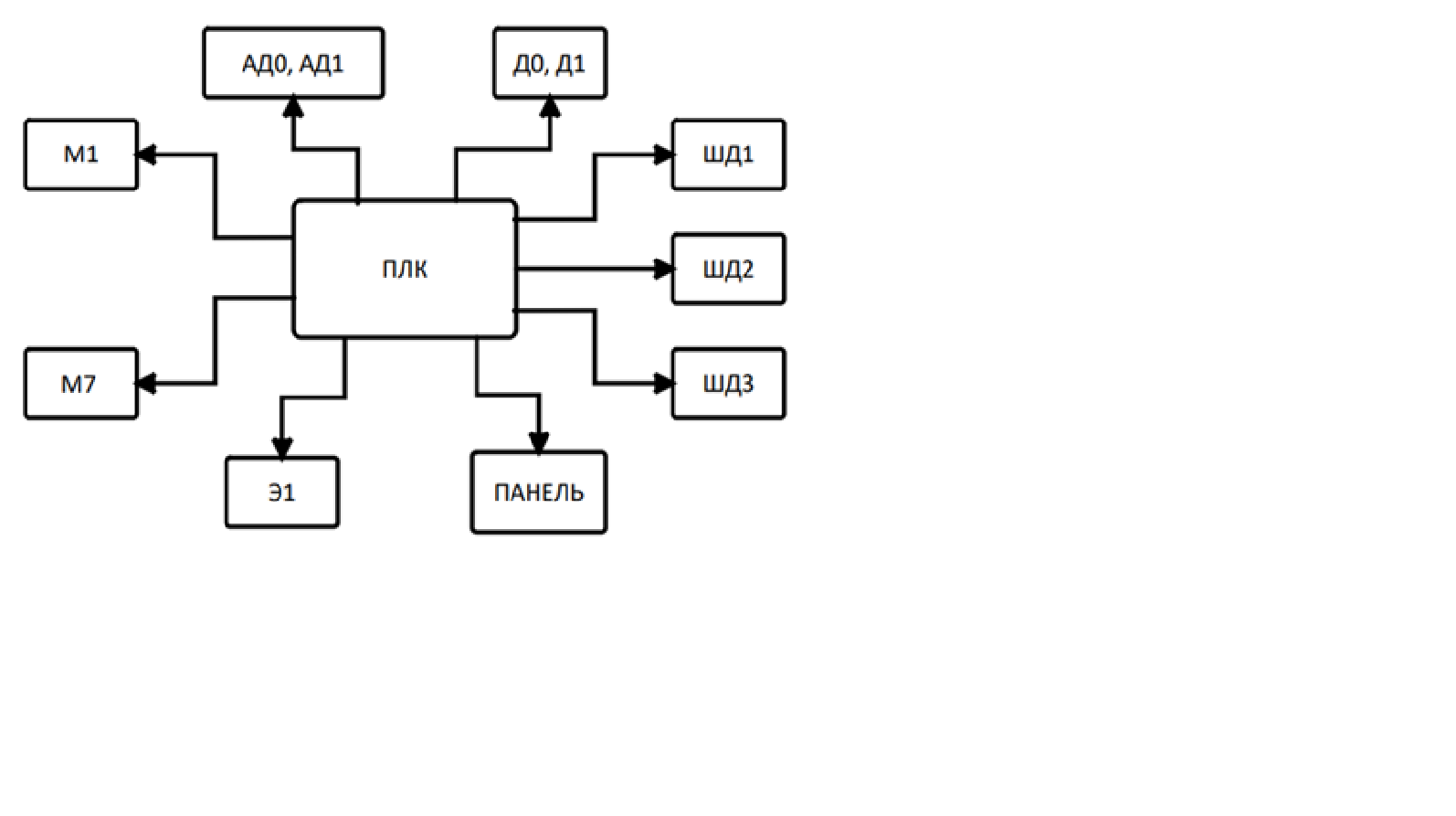


Рисунок 4 – упрощенное отображение электрических соединений ключевых компонентов системы управления станком.

На представлена схема, иллюстрирующая взаимосвязь между программными модулями (реализованными в разработанном ПО) и аппаратными компонентами системы управления станком с ЧПУ. Она показывает, какие программные модули отвечают за управление конкретными двигателями, датчиками и другими устройствами станка.

Описание компонентов схемы:

* Вх.дан. (Входные данные): Представляют собой блок, отвечающий за чтение данных из внешних источников (например, из файла, USB-накопителя). Эти данные содержат параметры обработки, траектории движения инструмента и другие необходимые параметры. Этот блок связан с:
  + RMP: модуль чтения параметров обработки.
  + Read\_Prog: Модуль чтения управляющей программы.
  + RMPS: Модуль чтения параметров для нескольких проходов.
* ШД1, ШД2, ШД3: Шаговые двигатели, отвечающие за перемещение по разным осям станка и двигатель ХВ.[5]
* M1: Двигатель шпинделя (токарной обработки).
* M7: Двигатель фрезы.
* Панель: Панель оператора, через которую пользователь взаимодействует с системой управления.
* Вых.дан. (Выходные данные): Блок, отвечающий за запись информации о ходе процесса и результате обработки. Связан с:
  + WMP – модуль записи параметров обработки.

Полное описание схемы:

1. Входные данные (Вх.дан.): Модуль отвечает за загрузку параметров обработки и управляющей программы. Информация передается на ПЛК для выполнения токарных и фрезерных операций. Подмодули: RMP, Read\_Prog и RMPS.

Движение каретки по контуру заготовки описывается следующим образом в :

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| **№ символа п/п** | **Возможное значение** | **Назначение** | **Описание** |
| 1 | 0 или 1 | Направление движения по оси «Y» | 0 – движение «назад» (от центра)  1 – движение «вперед» (к центру) |
| 2,3 | 01, 02, 03, 04, 05 | Перемещение по оси «Y» | в десятых долях мм |
| 4 | ; | Конец данных для текущего шага |  |
|  | \* | Конец данных для всей детали | Конец файла |

Пример

103; - перемещение вперед на 0,3мм

1. Управление двигателями (ШД1, ШД2, ШД3, M1, M7) [5]:
   * Шаговые двигатели (ШД1, ШД2, ШД3) управляются в соответствии с режимом работы. Модуль Passage1, 2 отвечает за формирование сигналов для шаговых двигателей при токарной обработке (однопроходной или двухпроходной).
   * Двигатель шпинделя (M1) и двигатель фрезы (M7) управляются модулем Milling для выполнения операций фрезерования [9].
   * Перемещения шаговых двигателей в нулевое положение, в т.ч. задание общих параметров, реализуются под управлением модуля M\_C, Zerro.
2. Панель оператора (Панель): Оператор через панель может выбирать режимы работы, задавать параметры обработки и контролировать процесс выполнения программы. Управление панелью оператора также осуществляется под общим руководством модуля M\_C, Zerro.
3. Выходные данные (Вых.дан.): Модуль записывает данные о ходе процесса в выходные файлы. Выходные данные управляются модулем WMP.



Рисунок 5 – взаимосвязь программных модулей с аппаратными компонентами станка.

## Разработка алгоритмов работы для каждого функционального модуля, обеспечивающих выполнение заданных операций

При разработке функциональных модулей необходимо разработать алгоритмы для каждого модуля, описывающие последовательность действий, необходимых для выполнения его функций.

**Примеры алгоритмов:**

* **Модуль сбора данных:**

1. Инициализация всех глобальных параметров, входных данных с USB-накопителя или с панели оператора.
2. Проверка на наличие ошибок при чтении данных.
3. Если произошла ошибка, закрытие файлов и переход станка в режим «ожидания».
4. Передача данных в модуль обработки данных.

* **Модуль обработки данных:**
  1. Получение данных от модуля сбора данных.
  2. Калибровка данных с использованием калибровочных коэффициентов.
  3. Вычисление производных значений.
  4. Проверка состояние данных и отклика станка.
  5. Передача обработанных данных в модуль принятия решений.
* **Модуль принятия решений:**
  1. Получение обработанных данных от модуля обработки данных.
  2. Сравнение данных с заданными пороговыми значениями.
  3. Определение необходимого действия в зависимости от результатов сравнения.
  4. Формирование управляющих сигналов.
  5. Передача управляющих сигналов в модуль управления.
* **Модуль управления:**

1. Получение обработанных данных от модуля принятия решений.
2. Распределение задач в зависимости от управляющих сигналов.
3. Проверка состояния станка.

**Методы проектирования алгоритмов:**

* **Блок-схемы:** визуальное представление алгоритма в виде блоков, соединенных стрелками.
* **Псевдокод:** описание алгоритма на естественном языке с использованием элементов синтаксиса языка программирования.

Алгоритм работы станка представлен на и Рисунок 7, программа должна состоять из следующих подпрограмм и обеспечивать возможность выполнения перечисленных ниже функций:

**«Панель оператора»:**

* выбор программы работы.
* ввод дополнительных данных для работы.

**«1 проход»:**

* точная обработка заготовки при малой линейной скорости и максимальным съёмом материала.
* вращающаяся заготовка.

**«2 прохода»:**

* грубая обработка заготовки при высокой линейной скорости и максимальным съёмом материала.
* точная обработка заготовки при малой линейной скорости и минимальным съёмом материала.
* инструмент – пила.
* вращающаяся заготовка.

**«1 проход+фрезерование»**

* точная обработка заготовки при малой линейной скорости и максимальным съёмом материала.
* точная обработка заготовки при малой линейной скорости и минимальным съёмом материала.
* инструмент – пила или фрезер.
* вращающаяся или неподвижная заготовка.

**«2 прохода+фрезерование»:**

* грубая обработка заготовки при высокой линейной скорости и максимальным съёмом материала.
* точная обработка заготовки при малой линейной скорости и минимальным съёмом материала.
* инструмент – пила или фрезер.
* вращающаяся или неподвижная заготовка.

**«Нулевое положение»**

* возврат каретки в начальное положение («0»).
* неподвижная заготовка.

**«Фрезерование»**

* точная обработка заготовки при малой линейной скорости и минимальным съёмом материала.
* инструмент – пила или фрезер.
* вращающаяся или неподвижная заготовка.

**«Загрузка»**

* происходит загрузка таблицы с данными из USB-накопителя в память ПЛК.

**«Ручное управление»**

передвижения каретки по двум осям координат в ручном режиме.

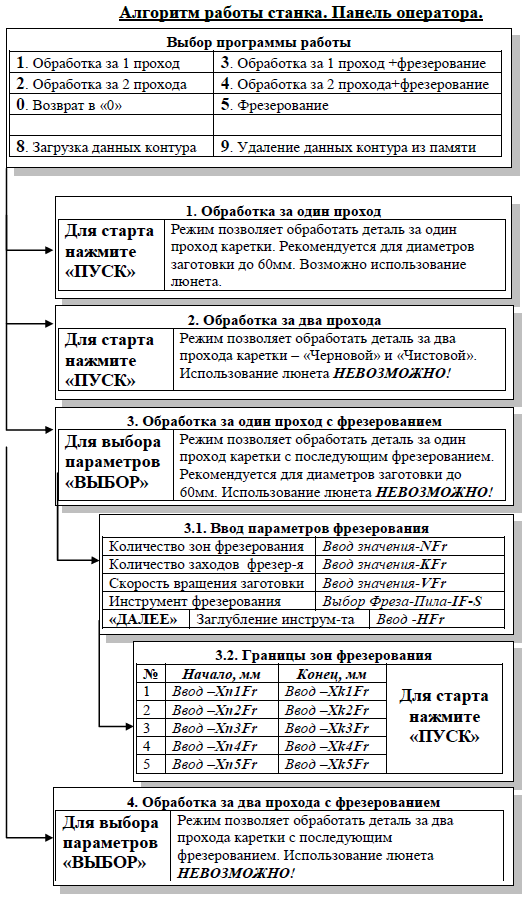


Рисунок 6 – алгоритм работы станка КТФ-8

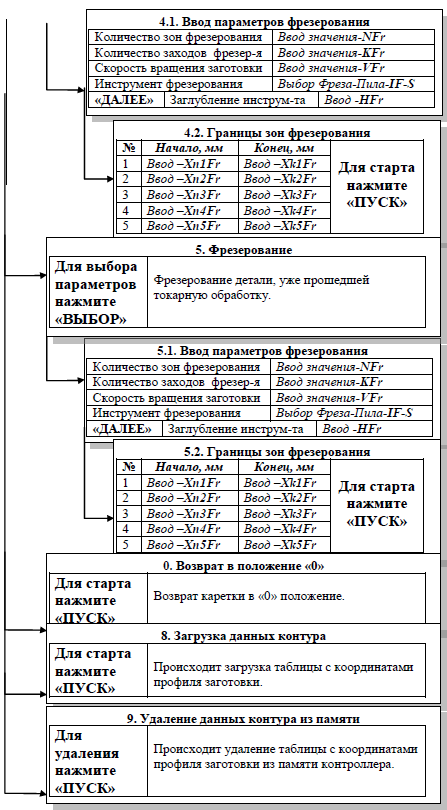


Рисунок 7 – алгоритм работы станка КТФ-8

## Разработка структуры данных для хранения и обработки информации, включая данные с датчиков, параметры обработки и управляющие сигналы

На этом этапе необходимо определить, какие структуры данных будут использоваться для хранения и передачи информации между модулями. Важно выбрать структуры данных, которые обеспечивают эффективный доступ к данным и удобство их обработки [3].

**Структуры данных [7] :**

* **Данные с датчиков:**
  + Массивы.
  + Переменные(логические, целочисленные, строковые).
* **Параметры обработки:**
  + Переменные(логические, целочисленные, строковые).
  + Глобальные переменные.
  + Функции.
* **Управляющие сигналы:**
  + Переменные(логические, целочисленные, строковые).

**Примеры реализации в CODESYS:**

В разработанном программном обеспечении широко используются различные подходы к обработке данных для обеспечения его функциональности и эффективности. В частности, реализацию одного из методов обработки входных данных можно детально изучить, обратившись к коду, представленному в . Этот пример демонстрирует конкретный алгоритм, применяемый для преобразования первичной информации, поступающей в систему, в формат, пригодный для дальнейшего анализа и использования.

Кроме того, важным аспектом работы ПО является использование результатов преобразования данных. В представлен пример функции [7], предназначенной для преобразования данных с последующим сохранением полученного результата в массиве. Такой подход позволяет организовать структурированное хранение обработанной информации, обеспечивая быстрый и удобный доступ к ней для других модулей и функций системы.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **// Функция для входных данных (преобразование двоичного представление в целое число)**  **FUNCTION BIN\_TO\_DWORD : WORD**  **VAR\_INPUT**  **BIN : STRING(40);**  **END\_VAR**  **VAR**  **pt : POINTER TO BYTE;**  **i: INT;**  **X: BYTE;**  **stop: INT;**  **END\_VAR** |

Листинг 1 – реализация функции преобразования двоичного представления данных в целое число.

|  |  |
| --- | --- |
| 1  2  3 | **// Массив данных**  **MAS[N\_C] := DEC\_TO\_BYTE(STR2); (\*Преобразуем строку в байт и помещаем в массив\*)** |

Листинг 2 – заполнение в массив входных данных.

Для обеспечения глобальной доступности данных и упрощения взаимодействия между различными компонентами разработанного ПО активно используются глобальные переменные. Описание структуры и назначения некоторых из этих переменных, предназначенных для использования во всей программе, а не в отдельных ее модулях, можно найти в . Использование глобальных переменных позволяет избежать многократной передачи одних и тех же данных между различными частями системы, повышая общую эффективность и упрощая структуру кода.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | **// Глобальные переменные**  **VAR\_GLOBAL**  **(\* Регистры сети ПЛК – Панель, RS-232 \*)**  **(\* Физические дискретные выходы \*)**  **Y1: BOOL; (\* Step ШД1, ось Y \*)**  **Y2: BOOL; (\* Dir ШД1, ось Y \*)**  **Y3: BOOL; (\* Step ШД3, ось X \*)**  **Y4: BOOL; (\*Step ШД2, вращение \*)**  **Y5 AT%QX3.0.0: BOOL; (\* Dir ШД2, вращение \*)**  **Y6 AT%QX3.0.1: BOOL; (\* Dir ШД3, ось X \*)**  **Y7 AT%QX3.0.2: BOOL; (\* М1 – мотор пилы \*)**  **Y8 AT%QX3.0.3: BOOL; (\* М7 – мотор фрезера \*)**  **…**  **state: BYTE := 65; (\* Номер шага в Read\_Prog \*)**  **N1: BYTE :=0; (\* Селектор в Stepper\_3\_Left \*)**  **N2: BYTE :=0; (\* Селектор в Stepper\_3\_Right \*)**  **R1: BYTE :=0; (\* Номер шага в Обработке за 1 проход \*)**  **R2: BYTE :=0; (\* Номер шага в Обработке за 2 прохода \*)**  **…**  **BIT\_16: BOOL; (\* Кнопка \*)**  **BIT\_17: BOOL; (\* Кнопка \*)**  **BIT\_18: BOOL; (\* Кнопка \*)**  **…**  **(\*Образы физических выходов \*)**  **M1: BOOL; (\* М1 – мотор пилы \*)**  **M7: BOOL; (\* М7 – мотор фрезера \*)**  **DIR\_2: BOOL; (\* Dir ШД2, вращение \*)**  **DIR\_3: BOOL; (\* Dir ШД3, ось X \*)**  **TON1: TON ;**  **END\_VAR**  **VAR\_GLOBAL RETAIN**  **END\_VAR** |

Листинг 3 – инициализация глобальных переменных.

## Выбор оптимальных языков программирования и библиотек CODESYS для реализации каждого модуля

CODESYS поддерживает несколько языков программирования стандарта IEC 61131-3, включая [3]:

* **LD (лестничная диаграмма):** графический язык, имитирующий электрические схемы. Удобен для логических операций и управления дискретными сигналами.
* **FBD (функционально-блочная диаграмма):** графический язык, основанный на функциональных блоках. Удобен для представления сложных алгоритмов обработки данных.
* **ST (структурированный текст):** текстовый язык, похожий на Pascal. Удобен для сложных вычислений и работы с массивами и структурами данных.
* **SFC (диаграмма последовательных функций):** графический язык для описания последовательности действий. Удобен для управления процессами, требующими четкой последовательности шагов.
* **IL (список инструкций):** низкоуровневый текстовый язык, похожий на ассемблер.

**Рекомендации по выбору языков:**

* **Модуль сбора данных:** ST.
* **Модуль обработки данных:** ST (для выполнения сложных вычислений и фильтрации данных).
* **Модуль принятия решений:** ST (для анализа данных и формирования управляющих сигналов), SFC (для реализации логических условий).
* **Модуль управления:** ST (для формирования аналоговых сигналов).
* **Модуль конфигурации:** ST (для чтения и записи файлов конфигурации)

**Библиотеки CODESYS [6] :**

CODESYS предоставляет широкий спектр библиотек для различных задач, вот описание каждой из перечисленных библиотек, используемых в среде CODESYS, и их назначение [3]:

* SysLibFile.lib 20.2.06: Эта библиотека предоставляет базовые функции для работы с файловой системой. Она позволяет выполнять операции чтения, записи, удаления файлов и каталогов, а также получать информацию о файлах (размер, дата создания и т. д.). Необходима для любых операций, связанных с хранением и обработкой данных на носителе (USB, встроенная память ПЛК).
* OwenLibFileAsync.lib 6.6.14: Эта библиотека является расширением SysLibFile.lib, предоставляющим асинхронные функции для работы с файлами. Асинхронный режим позволяет выполнять операции с файлами в фоновом режиме, не блокируя основной поток выполнения программы. Это особенно важно для приложений реального времени, где важна высокая отзывчивость. Библиотека предназначена для контроллеров ОВЕН и предоставляет специфические функции для работы с их файловой системой.
* SYSLIBTIME.LIB 20.2.06: Эта библиотека предоставляет функции для работы со временем и датой. Она позволяет получать текущее время, устанавливать время, выполнять операции с временными интервалами, форматировать дату и время. Необходима для ведения журнала, планирования задач, управления временем выполнения процессов.
* SYSTASKINFO.LIB 20.2.06: Эта библиотека предоставляет информацию о текущем состоянии системы, такую как использование памяти, загрузка процессора, количество задач и т. д. Полезно для мониторинга производительности и отладки приложений.
* SysLibPorts.lib 20.2.06: Эта библиотека обеспечивает доступ к портам ввода-вывода (цифровым и аналоговым) ПЛК. Позволяет считывать данные с датчиков и устройств, а также управлять исполнительными устройствами (реле, двигателями и т. д.).
* SYSLIBCALLBACK.LIB 20.2.06: Эта библиотека предоставляет механизмы для реализации функций обратного вызова (callbacks). Функции обратного вызова вызываются при наступлении определенных событий, таких как завершение асинхронной операции, изменение состояния датчика и т. д. Используется для организации событийного программирования и реактивного поведения системы.
* Timer.lib 13.12.08: Эта библиотека предоставляет функции для создания таймеров и управления ими. Таймеры позволяют выполнять определенные действия через заданные промежутки времени или по истечении определенной задержки. Используется для реализации периодических задач, задержек, сигнализации.
* PruAccessLib.lib 3.10.14: Эта библиотека предназначена для доступа к ресурсам, управляемым программируемыми блоками реального времени (PRU) на контроллерах ОВЕН. PRU представляют собой сопроцессоры, которые могут выполнять задачи в режиме реального времени с высокой точностью и скоростью. Используется для задач, требующих детерминированного поведения и минимальной задержки, например, для управления двигателями, обработки сигналов. Эта библиотека часто используется совместно с контроллерами ОВЕН, поддерживающими PRU.

В целом, эти библиотеки предоставляют широкий спектр функций, необходимых для разработки управляющих приложений для ПЛК в среде CODESYS. Они позволяют взаимодействовать с файловой системой, временем, портами ввода-вывода, событиями, таймерами и специализированными аппаратными ресурсами, что открывает широкие возможности для создания сложных и эффективных систем автоматизации.

## Реализация функциональных модулей программы в CODESYS

В этом разделе мы рассмотрим реализацию каждого функционального модуля программы в среде CODESYS. Особое внимание мы уделим выбору оптимальных языков программирования и библиотек, а также реализации логики управления и взаимодействия с оборудованием.

Панель оператора

**Цель:** настроить управление режимами работы, ввода параметров и отображения состояния системы на панели оператора ИП320.[1]

**Реализация:**

* **Логика обработки нажатий и ввода данных:**
  + Привязать переменные ПЛК к элементам управления на панели оператора.
  + Использовать события кнопок для обработки нажатий.
  + Реализовать проверку корректности введенных данных (диапазон значений, тип данных).
  + Установить значения соответствующих переменных ПЛК в зависимости от выбора пользователя.
* **Сохранение и загрузка параметров:**
  + Использовать функции работы с файлами CODESYS для сохранения параметров в файл на USB-накопителе (или во внутренней памяти ПЛК).
  + При запуске программы загружать параметры из файла.
  + При изменении параметров пользователем сохранять их в файл.

демонстрирует важную функцию разработанного программного обеспечения — чтение параметров фрезерования непосредственно с внешнего накопителя, вероятно, USB-накопителя, при нажатии кнопки «чтение/запись» на панели оператора. Рассмотрим подробно логику работы кода, его ключевые компоненты и предназначенные для них функции:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | **IF ((BIT\_34) AND (R10 = 120)) THEN R10 := 0; END\_IF**  **CASE R10 OF**  **0: (\* Сброс \*)**  **FL2 := FALSE; BIT\_4 := FALSE; BIT\_1 := FALSE; R10 := 3;**  **3: (\* Пауза \*)**  **TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)**  **IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R10 := 5; END\_IF**  **5: (\* Открытие файл на чтение, этап 1 \*)**  **(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)**  **res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));**  **IF res = ASYNC\_WORKING THEN**  **R10 := 10; (\* Нет ошибок, переход на открытие файла, этап 2 \*)**  **END\_IF**  **10:(\* Открытие файла на чтение, этап 2 \*)**  **(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)**  **res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));**  **IF res=ASYNC\_DONE THEN**  **IF handle <> 0 THEN**  **R10 := 20; (\* Нет ошибок, переход на чтение файла, этап 1 \*)**  **ELSE**  **R10 := 130;**  **END\_IF**  **ELSIF res < 0 THEN**  **R10 := 130;**  **END\_IF**  **20: (\* Чтение файла, этап 1 \*)**  **res:=OwenFileReadAsync(handle,ADR(L100), 240, ADR(result)); (\* Читаем 4 байта \*)**  **IF res = ASYNC\_WORKING THEN**  **R10 := 30; (\* Нет ошибок, переход на чтение файла, этап 2\*)**  **ELSE**  **R10:=40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)**  **END\_IF**  **30:(\* Чтение файла, этап 2\*)**  **res:=OwenFileReadAsync(handle, ADR(L100), 240, ADR(result));**  **IF res=ASYNC\_DONE THEN**  **IF result >= 0 THEN (\* Если нет ошибок, и количество считанных данных >=0, то перейти на закрытие файла \*)**  **R10 := 40;**  **ELSE**  **R10 := 40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)**  **END\_IF**  **ELSIF res < 0 THEN**  **R10 := 40;**  **END\_IF**  **40: (\* Закрытие файл, этап 1 \*)**  **res:=OwenFileCloseAsync(handle, ADR(result));**  **IF res=ASYNC\_WORKING THEN**  **R10 := 50; (\* Нет ошибок, переход на закрытие файла, этап 2 \*)**  **ELSE**  **R10 := 130; (\* \*)**  **END\_IF**  **50:(\* Закрытие файла, этап 2 \*)**  **res:=OwenFileCloseAsync(handle, ADR(result));**  **IF res=ASYNC\_DONE THEN**  **IF result = 0 THEN**  **R10 := 100; (\* \*)**  **ELSE**  **R10 := 130; (\* \*)**  **END\_IF**  **ELSIF res < 0 THEN**  **R10 := 130;**  **END\_IF**  **100:**  **L1 := MID (L100, 16, 1);**  **L2 := MID (L100, 16, 17);**  **L3 := MID (L100, 16, 33);**  **L4 := MID (L100, 16, 49);**  **L5 := MID (L100, 16, 65);**  **L6 := MID (L100, 16, 81);**  **L7 := MID (L100, 16, 97);**  **L8 := MID (L100, 16, 113);**  **L9 := MID (L100, 16, 129);**  **L10 := MID (L100, 16, 145);**  **L11 := MID (L100, 16, 161);**  **L12 := MID (L100, 16, 177);**  **L13 := MID (L100, 16, 193);**  **L14 := MID (L100, 16, 209);**  **L15 := MID (L100, 16, 225);**  **NFr := RG3 := BIN\_TO\_DWORD(L15);**  **KFr := RG4 := BIN\_TO\_DWORD(L14);**  **VFr := RG5 := BIN\_TO\_DWORD(L13);**  **HFr := RG6 := BIN\_TO\_DWORD(L12);**  **IF\_S := RG7 := BIN\_TO\_DWORD(L11);**  **Xn1Fr := RG8 := BIN\_TO\_DWORD(L10);**  **Xn2Fr := RG9 := BIN\_TO\_DWORD(L9);**  **Xn3Fr := RG10 := BIN\_TO\_DWORD(L8);**  **Xn4Fr := RG11 := BIN\_TO\_DWORD(L7);**  **Xn5Fr := RG12 := BIN\_TO\_DWORD(L6);**  **Xk1Fr := RG13 := BIN\_TO\_DWORD(L5);**  **Xk2Fr := RG14 := BIN\_TO\_DWORD(L4);**  **Xk3Fr := RG15 := BIN\_TO\_DWORD(L3);**  **Xk4Fr := RG16 := BIN\_TO\_DWORD(L2);**  **Xk5Fr := RG17 := BIN\_TO\_DWORD(L1);**  **R10 :=110;**  **110:(\* Файл загружен успешно \*)**  **FL2 := TRUE; BIT\_4 := TRUE; R10 :=120;**  **120:(\* Ожидание \*)**  **;**  **130: (\* Ошибка чтения параметров фреэерования \*)**  **BIT\_1 := TRUE; R10 :=120;**  **END\_CASE** |

Листинг 4 - чтение параметров фрезерования.

Описание кода:

* BIT\_34 – Логическая переменная, срабатывающая при нажатии кнопки в интерфейсе «Чтение/запись параметров фрезерования».
* R10 – Переменная состояния для данного автомата.

Рассмотрим подробно каждый шаг конечного автомата:

1. Состояние 0: Сброс (Reset)
   * Назначение: инициализация переменных и подготовка к процессу чтения.
   * Действия:
     + FL2 := FALSE;: Сброс флага, сигнализирующего об успешном считывании параметров.
     + BIT\_4 := FALSE;: Сброс флага, используемого для индикации состояния процесса.
     + BIT\_1 := FALSE;: Сброс флага, сигнализирующего об ошибке при чтении параметров.
     + R10 := 3;: Переход в состояние 3.
2. Состояние 3: Пауза (Pause)
   * Назначение: небольшая задержка для обеспечения готовности файловой системы к работе.
   * Действия:
     + TON1(IN := TRUE, PT:= UT\_1S);: Запуск таймера TON1 на 1 секунду.
     + IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R10 := 5; END\_IF: После завершения таймера сброс таймера и переход в состояние 5.
3. Состояние 5: Открытие файла для чтения, этап 1 (Open File, Step 1)
   * Назначение: асинхронный запуск операции открытия файла с параметрами.
   * Действия:
     + res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));: Вызов асинхронной функции для открытия файла.
       - NAME1: Имя файла, содержащего параметры фрезерования. Это может быть имя файла на USB-накопителе или в другом доступном месте.
       - 'r': Режим открытия файла «только для чтения».
       - ADR(handle): Указатель на переменную handle, в которую будет записан дескриптор открытого файла.
     + IF res = ASYNC\_WORKING THEN R10 := 10; END\_IFЕсли операция открытия файла успешно запущена и выполняется, автомат переходит в состояние 10.
4. Состояние 10: Открытие файла для чтения, этап 2 (Open File, Step 2)
   * Назначение: проверка завершения операции открытия файла и обработка результатов.
   * Действия:
     + res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));: Повторный вызов асинхронной функции открытия файла (для проверки статуса).
     + IF res=ASYNC\_DONE THEN: Если операция открытия файла завершена:
       - IF handle <> 0 THEN R10 := 20; ELSE R10 := 130; END\_IFЕсли файл успешно открыт (дескриптор файла handle не равен 0), автомат переходит в состояние 20. В противном случае (ошибка при открытии файла) автомат переходит в состояние 130.
     + ELSIF res < 0 THEN R10 := 130; END\_IF: Если при открытии файла (res < 0 ) произошла ошибка, автомат переходит в состояние 130.
5. Состояние 20: Чтение файла, этап 1 (Read File, Step 1)
   * Назначение: асинхронный запуск операции чтения данных из файла.
   * Действия:
     + res:=OwenFileReadAsync(handle, ADR(L100), 240, ADR(result));: Вызов асинхронной функции для чтения данных из файла.
       - handle: Дескриптор открытого файла.
       - ADR(L100): Указатель на буфер L100, в который будут считаны данные.
       - 240: Количество байт для чтения (размер буфера L100).
       - ADR(result): Указатель на переменную result, в которую будет записано количество фактически прочитанных байт.
     + IF res = ASYNC\_WORKING THEN R10 := 30; ELSE R10:=40; END\_IFЕсли операция чтения файла успешно запущена и выполняется, автомат переходит в состояние 30. В противном случае (ошибка чтения) автомат переходит в состояние 40.
6. Состояние 30: Чтение файла, этап 2 (Read File, Step 2)
   * Назначение: проверка завершения операции чтения файла и обработка результатов.
   * Действия:
     + res:=OwenFileReadAsync(handle, ADR(L100), 240, ADR(result));: Повторный вызов асинхронной функции чтения файла (для проверки статуса).
     + IF res=ASYNC\_DONE THEN: Если операция чтения файла завершена:
       - IF result >= 0 THEN R10 := 40; ELSE R10 := 40; END\_IFЕсли данные успешно прочитаны (result >= 0), автомат переходит в состояние 40. В противном случае (ошибка чтения) автомат переходит в состояние 40.
     + ELSIF res < 0 THEN R10 := 40; END\_IFЕсли при чтении файла (res < 0 произошла ошибка), автомат переходит в состояние 40.
7. Состояние 40: Закрытие файла, этап 1 (Close File, Step 1)
   * Назначение: асинхронный запуск операции закрытия файла.
   * Действия:
     + res:=OwenFileCloseAsync(handle, ADR(result));: Вызов асинхронной функции закрытия файла.
       - handle: Дескриптор открытого файла.
       - ADR(result): Указатель на переменную result, в которую будет записан код результата операции закрытия файла.
     + IF res=ASYNC\_WORKING THEN R10 := 50; ELSE R10 := 130; END\_IFЕсли операция закрытия файла выполнена успешно и находится в процессе выполнения, автомат переходит в состояние 50. В противном случае (ошибка закрытия) автомат переходит в состояние 130.
8. Состояние 50: Закрытие файла, этап 2 (Close File, Step 2)
   * Назначение: проверка завершения операции закрытия файла и обработка результатов.
   * Действия:
     + res:=OwenFileCloseAsync(handle, ADR(result));: Повторный вызов асинхронной функции закрытия файла (для проверки статуса).
     + IF res=ASYNC\_DONE THEN: Если операция закрытия файла завершена:
       - IF result = 0 THEN R10 := 100; ELSE R10 := 130; END\_IFЕсли файл успешно закрыт (result = 0), автомат переходит в состояние 100. В противном случае (ошибка при закрытии файла) автомат переходит в состояние 130.
     + ELSIF res < 0 THEN R10 := 130; END\_IFЕсли при закрытии файла (res < 0 ) произошла ошибка, автомат переходит в состояние 130.
9. Состояние 100: Анализ данных (Data Parsing)
   * Назначение: извлечение и преобразование параметров фрезерования из прочитанных данных.
   * Действия:
     + L1 := MID (L100, 16, 1); ... L15 := MID (L100, 16, 225);Извлечение отдельных строк из буфера L100 с помощью функции MID. Функция MID извлекает подстроку из строки, начиная с заданной позиции и заданной длины.
     + NFr := RG3 := BIN\_TO\_DWORD(L15); ... Xk5Fr := RG17 := BIN\_TO\_DWORD(L1);Преобразование извлеченных строк в числовые значения (DWORD — двойное слово, 32-разрядное целое число) с помощью функции BIN\_TO\_DWORD. Полученные числовые значения присваиваются переменным, которые используются для управления параметрами фрезерования:
       - NFr: Количество зон фрезерования
       - KFr: Количество заходов фрезы
       - VFr: Скорость вращения
       - HFr: Заглубление инструмента
       - IF\_S: Инструмент (фреза/пила/IF-S)
       - Xn1Fr - Xn5Fr: Начало зон фрезерования
       - Xk1Fr - Xk5Fr: Конец зон фрезерования
     + R10 :=110;: Переход в состояние 110.
10. Состояние 110: Файл загружен успешно (File Loaded Successfully)
    * Назначение: установка флагов, сигнализирующих об успешной загрузке параметров.
    * Действия:
      + FL2 := TRUE;: Установка флага успешного чтения параметров.
      + BIT\_4 := TRUE;: Установка флага, используемого для индикации состояния процесса.
      + R10 :=120;: Переход в состояние 120.
11. Состояние 120: Ожидание (Waiting)
    * Назначение: Ожидание новых действий пользователя.
    * Действия: Нет действий.
12. Состояние 130: Ошибка чтения параметров фрезерования (Error Reading Parameters)
    * Назначение: Обработка ошибки чтения параметров.
    * Действия:
      + BIT\_1 := TRUE;: Установка флага, сигнализирующего об ошибке при чтении параметров.
      + R10 :=120;: Переход в состояние 120.

Режимы обработки

**Цель:** Разработка алгоритмов управления движением каретки (ось X), вращением заготовки (ось Y) и мотора ходового винта в соответствии с заданными режимами обработки и входными данными.

**Реализация:**

* **Управление движением каретки (ось X), вращением заготовки (ось Y) и мотором ходового винта:**
  + Использовать шаговых двигателей ШД1 (каретка), ШД2 (заготовка) и ШД3(ходовой винт) [5].
  + Использовать функции управления двигателями CODESYS.
  + Задавать скорость, направление и позицию для каждого двигателя.
  + Контролировать текущее положение двигателей с помощью энкодера.
* **Реализация логики режимов обработки:**
  + Разработать отдельные функциональные блоки для каждого режима («1 проход», «2 прохода» и т.д.).
  + Внутри каждого блока реализовать логику управления движением каретки, вращением заготовки и ХВ в соответствии с выбранным режимом и данными из таблицы значений.
* **Контроль датчиков:**
  + Использование дискретных входов для контроля датчиков (Д0, Д1).
  + Реализовать логику обработки сигналов датчиков для обеспечения безопасности и правильной работы станка.
  + При срабатывании датчика остановить движение.

Код представленный в написанный на языке ST (структурированный текст) в среде CODESYS, представляет собой конечный автомат, который управляет процессом токарной обработки за один проход. Программа выполняет последовательность действий, включая выбор параметров, позиционирование инструмента, включение и выключение двигателей и перемещение по заданным координатам. Разберём код по частям, состояние за состоянием.

Глобальное описание:

Основная задача этого фрагмента кода — выполнить один проход токарной обработки заготовки на основе данных, загруженных в массив MAS. Процесс включает в себя заглубление инструмента, перемещение по оси X (продольная подача) и Y (подача на глубину), а также контроль различных параметров.

Переменные:

* R1: Целочисленная переменная, определяющая текущее состояние конечного автомата.
* BIT\_16: Логическая переменная, связанная с кнопкой на панели оператора или другим условием, запускающим процесс.
* FL1: Булева переменная, флаг, сигнализирующий об успешной загрузке данных из файла.
* S\_B\_A\_1.B0: Булева переменная, состояние чего-либо (концевого выключателя или датчика безопасности).
* SOM: Булева переменная («Start Of Machining» — начало обработки), флаг, сигнализирующий о начале процесса обработки.
* P\_N: Булева переменная («Номер прохода»), флаг, указывающий номер прохода.
* R100, R200: Целочисленные переменные, они связаны с регистрами или счётчиками.
* TON2: Таймер.
* UT\_2S: Константа времени таймера (2 секунды).
* FACTOR1: Вещественная переменная, вычисляемый коэффициент, связанный с параметрами углубления.
* P\_D1, P\_D3: Вещественные переменные, параметры.
* NUMBER\_I1, NUMBER\_I3: Целочисленные переменные, параметры.
* Kdr1, Kdr3: Вещественные переменные, параметры.
* STEP\_S\_Y, STEP\_S\_X: Вещественные переменные, параметры (шаги двигателей).
* Quantity\_3: DWord (двойное слово, 32-битное целое число без знака), количество импульсов для перемещения по оси X.
* Embedment: DWord, количество импульсов для углубления инструмента (ось Y).
* initial\_position\_Y, initial\_position\_X: Вызовы процедур или функциональных блоков для перемещения в исходное положение по осям Y и X.
* SSP\_Y, SSP\_X: Булевы переменные, флаги, сигнализирующие о достижении исходного положения по осям Y и X.
* M1: Логическая переменная, управляющая двигателем пилы.
* MAS: Массив, содержащий данные для обработки.
* N\_C: Счетчик, указывающий на текущий элемент в массиве MAS.
* N\_E: Индекс последнего элемента массива MAS, определяющий конец прохода.
* Stepper\_1, Stepper\_3: Вызов функциональных блоков управления шаговыми двигателями (ШД1 и ШД3).
* Task\_Speed, Accel, Quantity, Dir: Параметры для функциональных блоков управления шаговыми двигателями (скорость, ускорение, количество импульсов, направление).
* STOP\_STEPPER\_RUN\_STATE: Константа, представляющая состояние остановки шагового двигателя.
* SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_state, SteppersConfig\_Pru0MemoryTransfer.STEPPER3\_PRU0\_stepper\_state: Переменные, содержащие состояние шаговых двигателей. Используется общая память для обмена информацией между разными PRU (программируемыми блоками реального времени).
* Dir\_3: Логическая переменная, определяющая направление вращения ШД3.
* FL\_STEP1, FL\_STEP3: Флаги, показывающие, что шаговые двигатели завершили перемещение
* HD2\_1000, HD3\_STOP: Вызовы процедур или функциональных блоков для управления скоростью вращения заготовки и остановки двигателя.
* Step\_TIME: Это время шага (импульса) шагового двигателя.

Разбор состояний:

* 0: (\* Ожидание \*)
  + Программа находится в режиме ожидания, пока не будут выполнены следующие условия:
    - BIT\_16 = TRUE: Кнопка/сигнал запуска активированы.
    - FL1 = TRUE: Данные успешно загружены из файла.
    - NOT S\_B\_A\_1.B0: Состояние безопасности (концевой выключатель или датчик не активны, что сигнализирует о безопасности запуска).
    - NOT SOM: Флаг SOM должен быть установлен в значение FALSE, что означает, что процесс обработки еще не запущен.
  + Если все условия выполнены:
    - Stop\_Motor(): Вызывается функция остановки двигателя.
    - P\_N := TRUE: Устанавливается флаг, указывающий на то, что это первый проход (P\_N — «Номер прохода»).
    - SOM := TRUE: Устанавливается флаг SOM, сигнализирующий о начале процесса обработки.
    - R100 := 0; R200 := 0
    - TON2(IN := FALSE, PT:= T#0s);: Сбрасывается таймер TON2.
    - FACTOR1 := (P\_D1 \* NUMBER\_I1 \* Kdr1)/STEP\_S\_Y;: Вычисляется значение переменной FACTOR1, которое используется для расчёта углубления инструмента.
    - Quantity\_3 := REAL\_TO\_DWORD((P\_D3 \* NUMBER\_I3 \* Kdr3)/STEP\_S\_X);: Вычисляется значение переменной Quantity\_3, которое используется для задания перемещения по оси X.
    - R1 := 5: Программа переходит в состояние 5.
* 5: (\* Исходное положение Y \*)
  + initial\_position\_Y;: Вызывается функция или функциональный блок для перемещения инструмента в исходное положение по оси Y (глубина).
  + IF (SSP\_Y = TRUE ) THEN R1 := 10; END\_IF: Если достигнуто исходное положение по оси Y (сигнал SSP\_Y = TRUE), программа переходит в состояние 10.
* 10: (\* Исходное положение X \*)
  + initial\_position\_X;: Вызывается функция или функциональный блок для перемещения инструмента в исходное положение по оси X (продольное).
  + IF (SSP\_X = TRUE ) THEN R1 := 20; END\_IF: Если достигнуто исходное положение по оси X (сигнал SSP\_X = TRUE), программа переходит в состояние 20.
* 20: ( Включаем мотор пилы и быстрое вращение заготовки)\*\*
  + M1 := TRUE; R1 := 30;: Двигатель M1 (пилы) включается. Программа переходит в состояние 30.
* 30: (\* Пауза 2 секунды \*)
  + TON2(IN := TRUE, PT:= UT\_2S);: Запускается таймер TON2 на 2 секунды.
  + IF (TON2.Q=TRUE) THEN TON2(IN := FALSE, PT:= T#0s); Embedment := REAL\_TO\_DWORD(FACTOR1 \* 100 \* MAS[0]); R1 := 35; END\_IF: После истечения 2 секунд:
    - Таймер сбрасывается.
    - Вычисляется значение Embedment на основе FACTOR1, 100 и первого элемента массива MAS[0]. Embedment определяет глубину погружения инструмента.
    - Программа переходит в состояние 35.
* 35: (\* Углубление в заготовку на начальном диаметре \*)
  + Stepper\_1(Task\_Speed := 55000, Accel := 50000, Quantity := Embedment, Dir := FALSE);: Вызывается функциональный блок Stepper\_1 для управления шаговым двигателем (ШД1), отвечающим за перемещение по оси Y (глубина).
    - Task\_Speed := 55000: Задается скорость перемещения.
    - Accel := 50000: Задается ускорение.
    - Quantity := Embedment: Задается количество импульсов для перемещения на глубину Embedment.
    - Dir := FALSE: Задается направление перемещения (вероятно, заглубление).
  + IF SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_state = STOP\_STEPPER\_RUN\_STATE THEN SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_enable := FALSE; N\_C := 1; R1 := 40; END\_IF Когда ШД1 достигнет заданного положения (т.е. будет остановлен), двигатель отключается, устанавливается N\_C := 1, и программа переходит в состояние 40. N\_C начинает отсчет с 1, поскольку MAS[0] уже был использован для первоначального углубления.
* 40: (\*Включаем ШД3 и ШД1\*)
  + Stepper\_3(EN := TRUE, RES := FALSE, PERIOD := 100, DURABILITY := 50); Dir\_3 := TRUE;: Включается шаговый двигатель ШД3, отвечающий за перемещение по оси X (продольная подача).
    - EN := TRUE: Разрешение работы двигателя.
    - RES := FALSE: Отсутствие сброса (RESET).
    - PERIOD := 100: Период импульсов (определяет скорость).
    - DURABILITY := 50: Длительность импульса (скважность).
    - Dir\_3 := TRUE: Устанавливается направление движения по оси X
  + Закомментированный код:
    - Направление (Dir\_1) и количество импульсов (Quantity\_1) для ШД1 (подача на глубину) вычислялись на основе текущего элемента массива MAS[N\_C].
    - ШД1 запускался с заданными параметрами.
  + FL\_STEP1 := FALSE; FL\_STEP3 := FALSE;: Сбрасываются флаги завершения перемещения шаговых двигателей
  + R1 := 45: Программа переходит в состояние 45.
* 45: (\* Отмеряем перемещения по осям X и Y \*)
  + IF (FL\_STEP3 = TRUE) THEN N\_C := N\_C + 1; R1 := 50; END\_IF Если ШД3 (ось X) завершил перемещение, увеличиваем счетчик N\_C и переходим в состояние 50.
* 50: (\* Анализ конца прохода \*)
  + IF (N\_C > N\_E) THEN R1 := 60; ELSE R1 := 40; END\_IF Если счетчик N\_C превышает индекс последнего элемента массива N\_E, значит, все данные из массива MAS обработаны и проход завершен. Программа переходит в состояние 60. В противном случае программа возвращается в состояние 40 для обработки следующей точки.
* 60: (\* Выключаем М1, ШД2, М3 \*)
  + M1 := FALSE; HD3\_STOP; SOM := FALSE; R200 := 0; R1 := 70;: Двигатель M1, вращение, флаг SOM деактивируются, R200 обнуляется, и программа переходит в состояние 70.
* 70: (\* Переводим инструмент в нулевое положение \*)
  + initial\_position\_Y;: Вызывается функция или функциональный блок для перемещения инструмента в исходное положение по оси Y (глубина).
  + IF (SSP\_Y = TRUE) THEN SOM := FALSE; R1 := 0; END\_IF: Если достигнуто исходное положение по оси Y (сигнал SSP\_Y = TRUE), флаг SOM деактивируется, и программа переходит в состояние 0 (ожидание), готовая к следующему проходу.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 | **(\* Обработка за 1 проход \*)**  **CASE R1 OF**  **0: (\* Ожидание \*)**  **IF ((BIT\_16) AND (FL1 = TRUE) AND (NOT S\_B\_A\_1.B0) AND (NOT SOM)) THEN**  **Stop\_Motor();**  **P\_N := TRUE; (\* Номер прохода 1 \*)**  **SOM := TRUE;**  **R100 := 0;**  **R200 := 0;**  **TON2(IN := FALSE, PT:= T#0s);**  **FACTOR1 := (P\_D1 \* NUMBER\_I1 \* Kdr1)/STEP\_S\_Y;**  **Quantity\_3 := REAL\_TO\_DWORD((P\_D3 \* NUMBER\_I3 \* Kdr3)/STEP\_S\_X); (\* Число импульсов, которое подается на драйвер ШД3 для перемещения по Х на 0,1 мм \*)**  **R1 := 5;**  **END\_IF**  **5: (\* Исходное положение Y \*)**  **initial\_position\_Y;**  **IF (SSP\_Y = TRUE ) THEN R1 := 10; END\_IF**  **10: (\* Исходное положение X \*)**  **initial\_position\_X;**  **IF (SSP\_X = TRUE ) THEN R1 := 20; END\_IF**  **20: (\* Включаем мотор пилы и быстрое вращение заготовки\*)**  **(\*M1 := TRUE; HD2\_1000; R1 := 30;\*)**  **M1 := TRUE; R1 := 30;**  **30: (\* Пауза 2 сек \*)**  **TON2(IN := TRUE, PT:= UT\_2S); (\* Запустили таймер \*)**  **IF (TON2.Q=TRUE) THEN TON2(IN := FALSE, PT:= T#0s); Embedment := REAL\_TO\_DWORD(FACTOR1 \* 100 \* MAS[0]); R1 := 35; END\_IF (\* 1 в MAS[0] означает углубление на 1 мм, 2 - на 2мм и т.д. \*)**  **35: (\* Углубление в заготовку на начальный диаметр \*)**  **Stepper\_1(Task\_Speed := 55000, Accel := 50000, Quantity := Embedment, Dir := FALSE);**  **IF SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_state = STOP\_STEPPER\_RUN\_STATE THEN**  **SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_enable := FALSE; N\_C := 1; R1 := 40; END\_IF**  **40: (\* Включаем ШД3 и ШД1 \*)**  **Stepper\_3(EN := TRUE, RES := FALSE, PERIOD := 100, DURABILITY := 50); Dir\_3 := TRUE;**  **(\***  **IF ((MAS[N\_C]/100) = 0) THEN Dir\_1 := TRUE; (\* Вперед \*)**  **ELSE Dir\_1 := FALSE; END\_IF (\* Назад \*)**  **Quantity\_1 := REAL\_TO\_DWORD(FACTOR1 \* (MAS[N\_C] MOD 100));**  **Accel\_1 := REAL\_TO\_DWORD((2 \* Quantity\_1)/(Step\_TIME \* Step\_TIME));**  **Stepper\_1(Task\_Speed := 300000, Accel := Accel\_1, Quantity := Quantity\_1, Dir := Dir\_1);**  **\*)**  **FL\_STEP1 := FALSE; FL\_STEP3 := FALSE;**  **R1 := 45;**  **45: (\* Отмеряем перемещения по Х и Y \*)**  **(\*IF SteppersConfig\_Pru0MemoryTransfer.STEPPER3\_PRU0\_stepper\_state = STOP\_STEPPER\_RUN\_STATE THEN (\* Если отработано перемещение по X \*)**  **SteppersConfig\_Pru0MemoryTransfer.STEPPER3\_PRU0\_stepper\_enable := FALSE; FL\_STEP3 := TRUE; END\_IF\*)**  **;**  **(\***  **IF SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_state = STOP\_STEPPER\_RUN\_STATE THEN (\* Если отработано перемещение по Y \*)**  **SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_enable := FALSE; FL\_STEP1 := TRUE; END\_IF**  **IF ((FL\_STEP1 = TRUE) AND (FL\_STEP3 = TRUE)) THEN N\_C := N\_C + 1; R1 := 50; END\_IF**  **\*)**  **IF (FL\_STEP3 = TRUE) THEN N\_C := N\_C + 1; R1 := 50; END\_IF**  **50: (\* Анализ конца прохода \*)**  **IF (N\_C > N\_E) THEN R1 := 60; ELSE R1 := 40; END\_IF**  **60: (\* Выключаем М1, ШД2, М3 \*)**  **M1 := FALSE; HD3\_STOP; SOM := FALSE; R200 := 0; R1 := 70;**  **70: (\* Переводим инструмент в нулевое положение \*)**  **initial\_position\_Y;**  **(\*initial\_position\_X;\*)**  **IF (SSP\_Y = TRUE) THEN SOM := FALSE; R1 := 0; END\_IF**  **END\_CASE** |

Листинг 5 – обработка изделия за 1 проход.

Режим «Нулевое положение»

**Цель:** разработка алгоритма возврата каретки в исходное положение (координата 0) с использованием как скоростного, так и точного позиционирования.

**Реализация:**

* **Управление двигателями ХВ и перемещения каретки:**
  + Использовать функции управления двигателями CODESYS (как и в режимах обработки).
  + Начать движение каретки к нулевому положению.
  + При приближении к нулевому положению (на основе данных с энкодера) снизить скорость для точного позиционирования.
* **Контроль датчиков:**
  + Используйте датчики (Д0, Д1) для точного определения нулевого положения.
  + Остановить движение при срабатывании датчика.

Код, предназначенный для перевода кареток станка в нулевое (исходное) положение при нажатии кнопки на панели управления, предоставлен в . Он использует конечный автомат и имеет отдельную логику для осей X и Y.

Ключевые переменные и элементы:

* R7: Переменная состояния, управляющая логикой конечного автомата.
* BIT\_27Логическая переменная, связанная с кнопкой на панели управления, активирующей режим перевода в нулевое положение.
* SOMЛогическая переменная, указывающая на то, что процесс перевода в нулевое положение активен.
* Stop\_Motor(): Функция для остановки моторов.
* BIT\_10Логическая переменная, которая, вероятно, используется для индикации процесса поиска нулевого положения (например, для включения светодиода).
* SSP\_X, SSP\_Y: Логические переменные, сигнализирующие о достижении исходного положения по осям X и Y соответственно (TRUE — достигнуто, FALSE — не достигнуто).
* initial\_position\_YФункция, реализующая движение к исходному положению по оси Y.
* initial\_position\_XФункция, реализующая движение к исходному положению по оси X.
* BIT\_36Логическая переменная, связанная с кнопкой «SET» (вероятно, для принудительной остановки процесса и установки текущего положения в нулевое).
* HD3\_STOPФункция для остановки шагового двигателя 3 (вероятно, шагового двигателя, отвечающего за перемещение по оси Y).

Описание состояний конечного автомата:

* 0: (\* Ожидание \*)
  + Назначение: ожидание нажатия кнопки перевода в нулевое положение.
  + Действия:
    - IF (BIT\_27 AND (NOT SOM)) THEN ... END\_IF: Этот условный оператор проверяет:
      * BIT\_27: Кнопка возврата в нулевое положение нажата.
      * (NOT SOM): Процесс перевода в нулевое положение еще не завершен.
    - Если оба условия выполнены:
      * Stop\_Motor();: Остановка моторов.
      * BIT\_10 := TRUE;: Включение индикации процесса поиска нулевого положения.
      * SSP\_X := FALSE;: Сброс флага достижения исходного положения по оси X.
      * SSP\_Y := FALSE;: Сброс флага достижения исходного положения по оси Y.
      * SOM := TRUE;: Установка флага активации процесса перевода в нулевое положение.
      * R7 := 10;: Переход в состояние 10.
* 10: (\* Исходное положение по оси Y \*)
  + Назначение: перемещение каретки в исходное положение по оси Y.
  + Действия:
    - IF (SSP\_Y = FALSE) THEN initial\_position\_Y; ELSE R7 := 40; END\_IF:
      * Если исходное положение по оси Y еще не достигнуто (SSP\_Y = FALSE), вызывается функция initial\_position\_Y для перемещения каретки.
      * В противном случае, если исходное положение по оси Y уже достигнуто, происходит переход к состоянию 40.
* 40: (\* Точное позиционирование в нулевое положение по оси X \*)
  + Назначение: перемещение каретки в исходное положение по оси X.
  + Действия:
    - IF (SSP\_X = FALSE) THEN initial\_position\_X; END\_IFЕсли исходное положение по оси X еще не достигнуто (SSP\_X = FALSE), вызывается функция initial\_position\_X для перемещения каретки.
    - IF (SSP\_X) THEN BIT\_10 := FALSE; SOM := FALSE; R7 := 0; END\_IFЕсли исходное положение по оси X достигнуто (SSP\_X = TRUE), происходит следующее:
      * Выключение индикации процесса поиска нулевого положения (BIT\_10 := FALSE).
      * Сброс флага активности процесса перевода в нулевое положение (SOM := FALSE).
      * Переход в состояние 0 (ожидание).
    - IF (BIT\_36) THEN HD3\_STOP; R7 := 0; BIT\_10 := FALSE; SOM := FALSE; END\_IFЕсли нажата кнопка «SET» (BIT\_36 = TRUE), происходит следующее:
      * Остановка шагового двигателя (HD3\_STOP).
      * Переход в состояние 0 (ожидание).
      * Выключение индикации процесса поиска нулевого положения (BIT\_10 := FALSE).
      * Сброс флага активности процесса перевода в нулевое положение (SOM := FALSE).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **(\* Нулевое положение, перевод кареток в исходное положение по нажатию кнопки на панели \*)**  **CASE R7 OF**  **0: (\* Ожидание \*)**  **IF (BIT\_27 AND (NOT SOM)) THEN**  **Stop\_Motor();**  **BIT\_10 := TRUE;**  **SSP\_X := FALSE; (\* Нет исходного положения \*)**  **SSP\_Y := FALSE; (\* Нет исходного положения \*)**  **SOM := TRUE;**  **R7 := 10;**  **END\_IF**  **10: (\* Исходное положение по оси Y \*)**  **IF (SSP\_Y = FALSE) THEN initial\_position\_Y; ELSE R7 := 40;END\_IF (\* Ищим нулевое положение по оси Y \*)**  **40: (\* Точное позиционирование в нулевое положение по оси Х \*)**  **IF (SSP\_X = FALSE) THEN initial\_position\_X; END\_IF**  **IF (SSP\_X) THEN BIT\_10 := FALSE; SOM := FALSE; R7 := 0; END\_IF (\* Ели встали в нулевое положение \*)**  **IF (BIT\_36) THEN HD3\_STOP; R7 := 0; BIT\_10 := FALSE; SOM := FALSE; END\_IF (\* Ели нажали кнопку SET \*)**  **END\_CASE** |

Листинг 6 – перевод каретки в «нулевое положение»

Режим «Фрезерование»

**Цель:** Разработка алгоритма фрезерной обработки с учётом параметров фрезерования (количество зон, количество проходов, скорость вращения заготовки, глубина погружения инструмента).

**Реализация:**

* **Управление двигателем фрезера (М7):**
  + Использовать дискретный выход для включения/выключения двигателя фрезера.
  + Может потребоваться управление частотой вращения двигателя.
* **Управление вращением заготовки (ШД2):**
  + Использовать шагового двигателя ШД2 для вращения заготовки.
  + Реализовать управление скоростью вращения.
* **Расчет траектории инструмента:**
  + Разработать алгоритмы расчета траектории инструмента для каждой зоны фрезерования.
  + Учитывать параметры фрезерования (количество зон, количество проходов, глубина погружения).
  + Используйте математические функции CODESYS для вычисления координат.
  + Сформировать последовательность команд для управления движением каретки (ось X) и вращением заготовки (ось Y) в соответствии с рассчитанной траекторией.

демонстрирует управление процессом фрезерования на станке с ЧПУ. Он также использует конечный автомат для организации последовательности действий.

Ключевые переменные и элементы:

* R5: Переменная состояния, управляющая логикой конечного автомата.
* BIT\_20: Логическая переменная, вероятно, связанная с кнопкой или переключателем, активирующим режим фрезерования.
* FL2: Логическая переменная, сигнализирующая об успешной загрузке параметров (вероятно, параметров фрезерования).
* S\_B\_A\_1.B0: Логическая переменная, вероятно, связанная с датчиком или переключателем, сигнализирующим о правильной настройке оборудования (например, о положении рычага).
* SOM: Логическая переменная, сигнализирующая о выполнении каких-либо действий (например, запуске фрезерования).
* Stop\_Motor(): Функция для остановки мотора.
* P\_N: Логическая переменная, указывающая номер прохода (в данном случае 1).
* RG23: Переменная, используемая для управления отображением сообщения (вероятно, в пользовательском интерфейсе).
* C: Переменная, представляющая показания энкодера (положение по оси).
* SSP\_Y: Логическая переменная, сигнализирующая о достижении исходного положения по оси Y.
* initial\_position\_Y: Функция, реализующая движение к исходному положению по оси Y.
* IF\_S: Переменная, указывающая тип используемого инструмента (0 — фреза, 1 — пила).
* M7, M1: Логические переменные, управляющие включением соответствующих инструментов (M7 — фреза, M1 — пила).
* M3\_1(F := 100, D := FALSE) Функция управления мотор-редуктором по оси X, где:
  + F: Скорость (частота).
  + D: Направление вращения (FALSE — вперед, TRUE — назад).
* Xn1Fr: Переменная, представляющая координату начала первого участка фрезерования.

Описание состояний конечного автомата:

* 0: (\* Ожидание \*)
  + Назначение: ожидание условий для запуска процесса фрезерования.
  + Действия:
    - IF (BIT\_20 AND FL2 = TRUE AND (NOT S\_B\_A\_1.B0) AND (NOT SOM)) THEN ... END\_IF Этот условный оператор проверяет следующие условия:
      * BIT\_20: Активирован режим фрезерования.
      * FL2 = TRUE: Параметры фрезерования успешно загружены.
      * (NOT S\_B\_A\_1.B0): Все условия безопасности соблюдены, рычаг находится в правильном положении.
      * (NOT SOM): Фрезерование еще не началось.
    - Если все условия выполнены:
      * Stop\_Motor();: Остановка мотора.
      * P\_N := TRUE;: Установка номера прохода 1.
      * RG23 := 1;: Скрывает сообщение «ПЕРЕВЕДИТЕ РЫЧАГ В ПОЛОЖЕНИЕ ДЛЯ ФРЕЗЕРОВАНИЯ».
      * C := 0;: Обнуление показаний энкодера.
      * SSP\_Y := FALSE;: Сброс флага достижения исходного положения по оси Y.
      * SOM := TRUE;: Установка флага начала фрезерования.
      * R5 := 10;: Переход в состояние 10.
* 10: (\* Исходное положение \*)
  + Назначение: перемещение инструмента в исходное положение по оси Y.
  + Действия:
    - initial\_position\_Y;: Вызов функции для перемещения в исходное положение по оси Y.
    - IF (SSP\_Y) THEN R1 := 20; END\_IF: Если исходное положение по Y достигнуто (SSP\_Y = TRUE), переход в состояние 20.
* 30: (\* Выбор инструмента для фрезерования \*)
  + Назначение: активация соответствующего инструмента (фрезы или пилы).
  + Действия:
    - IF (IF\_S = 0) THEN M7 := TRUE; R5 := 40; ELSE M1 := TRUE; R5 := 40; END\_IF:
      * Если IF\_S = 0 (выбрана фреза), активируется инструмент M7.
      * Если IF\_S = 1 (выбрана пила), активируется инструмент M1.
    - Переход в состояние 40.
* 40: (\* Включаем мотор-редуктор привода ХВ — вращение вперёд \*)
  + Назначение: запуск мотор-редуктора для перемещения по оси X.
  + Действия:
    - M3\_1(F := 100, D := FALSE);: Вызов функции для запуска мотор-редуктора со скоростью 100 в прямом направлении.
    - R5 := 50;: Переход в состояние 50.
* 50: (\* Ищем первую точку начала фрезерования \*)
  + Назначение: Ожидание достижения координаты начала первого участка фрезерования.
  + Действия:
    - IF (C >= Xn1Fr) THEN R5 := 60; END\_IFЕсли текущая координата (C) больше или равна координате начала первого участка фрезерования (Xn1Fr), переход в состояние 60.
* 60: (\* Начало фрезерования \*)
  + Назначение: изменение скорости перемещения по оси X для выполнения фрезерования.
  + Действия:
    - M3\_1(F := 50, D := FALSE);: Вызов функции для изменения скорости мотор-редуктора на 50.
    - R5 := 70;: Переход в состояние 70.
* R5 := 0;: Переход в состояние 0.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **(\* Фрезерование \*)**  **CASE R5 OF**  **0: (\* Ожидание \*)**  **IF (BIT\_20 AND FL2 = TRUE AND (NOT S\_B\_A\_1.B0) AND (NOT SOM)) THEN**  **Stop\_Motor();**  **P\_N := TRUE; (\* Номер прохода 1 \*)**  **RG23 := 1; (\* Гасим сообщение ПЕРЕВЕДИТЕ РЫЧАГ В ПОЛОЖЕНИЕ ФРЕЗЕРОВАНИЯ \*)**  **C := 0; (\* Обнулили показания энкодера \*)**  **SSP\_Y := FALSE; (\* Нет исходного положения \*)**  **SOM := TRUE;**  **R5 := 10;**  **END\_IF**  **10: (\* Исходное положение \*)**  **initial\_position\_Y;**  **IF (SSP\_Y) THEN R1 := 20; END\_IF**  **(\*IF (NOT D4\_2) THEN BLINKER; OUT.3 := FALSE; OUT.1 := OUT\_B; SysPortOut(0, OUT); ELSE RG23 := 1; R5 := 30; END\_IF\*)**  **30: (\* Выбор инструмента для фрезерования \*)**  **IF (IF\_S = 0) THEN M7 := TRUE; R5 := 40; ELSE M1 := TRUE; R5 := 40; END\_IF**  **40: (\* Включаем мотор-редуктор привода ХВ - вращение вперёд \*)**  **M3\_1(F := 100, D := FALSE); R5 := 50;**  **50: (\* Ищем первую точку начала фрезерования \*)**  **IF (C >= Xn1Fr) THEN R5 := 60; END\_IF**  **60: (\* Начало фрезерования \*)**  **M3\_1(F := 50, D := FALSE); R5 := 70;**  **R5 := 0;**  **END\_CASE;** |

Листинг 7 – режим фрезерования.

Режим “Загрузка”

**Цель:** Разработка функции загрузки данных из текствого файла с USB-накопителя в память ПЛК [2].

**Реализация:**

* **Чтение файла:**
  + Использование функции работы с файлами CODESYS для открытия и чтения текстового файла с USB-накопителя.
  + Читать файл построчно.
* **Обработка данных:**
  + Разделить каждую строку на отдельные значения (используя разделитель, например, запятую).
  + Преобразовать значения в нужный тип данных (REAL, INT и т.д.).
  + Проверять формат данных на соответствие ожидаемому.
* **Хранение данных в памяти ПЛК:**
  + Создать структуру данных для хранения загруженных данных (например, ARRAY OF STRUCT).
  + Записывать преобразованные значения в эту структуру.
* **Обработка ошибок:**
  + Обработка ошибок при открытии файла, чтении данных, преобразовании типов и т.д.
  + Выводить сообщения об ошибках на панель оператора.

Код на представляет собой фрагмент программы, написанной на языке ST в среде CODESYS, которая отвечает за чтение данных из текстового файла, расположенного на USB-накопителе, и сохранение их в массиве. Программа работает по принципу конечного автомата, где переменная state определяет текущее состояние выполнения. Давайте разберём код по частям:

1. Выбор файла на USB-накопителе (Выбор программы)

IF ((BIT\_21) AND (state = 65)) THEN NAME := 'usb:p1.txt'; RG20 := 1; state := 3; END\_IF

IF ((BIT\_22) AND (state = 65)) THEN NAME := 'usb:p2.txt'; RG20 := 2; state := 3; END\_IF

IF ((BIT\_23) AND (state = 65)) THEN NAME := 'usb:p3.txt'; RG20 := 3; state := 3; END\_IF

IF ((BIT\_24) AND (state = 65)) THEN NAME := 'usb:p4.txt'; RG20 := 4; state := 3; END\_IF

IF ((BIT\_25) AND (state = 65)) THEN NAME := 'usb:p5.txt'; RG20 := 5; state := 3; END\_IF

IF ((BIT\_26) AND (state = 65)) THEN NAME := 'usb:p6.txt'; RG20 := 6; state := 3; END\_IF

* Этот блок кода выполняется, когда state равен 65 (состояние ожидания).
* Он проверяет, установлен ли какой-либо из битов BIT\_21 - BIT\_26. Каждый бит соответствует выбору определенного файла.
* Если один из битов установлен, то:
  + Переменной NAME присваивается имя соответствующего файла (например, 'usb:p1.txt'). Префикс usb: указывает на то, что файл находится на USB-накопителе.
  + Переменной RG20 присваивается номер выбранного файла.
  + state устанавливается на 3, что переводит программу в режим паузы.

2. Конечный автомат (Основная логика программы)

Разбор состояний:

* 3: (\* Пауза \*)
  + Запускается таймер TON1 на время UT\_1S.
  + После истечения времени, заданного таймером, программа переходит в состояние 5. Пауза, вероятно, нужна для того, чтобы дать USB-накопителю время на инициализацию.
* 5: (\* Открытие файла для чтения, этап 1 \*)
  + Функция OwenFileOpenAsync пытается открыть файл с именем, хранящимся в переменной NAME, для чтения ('r'). Функция является асинхронной, то есть не блокирует выполнение программы во время открытия файла.
  + ADR(handle) - Получает адрес переменной handle, в которой будет храниться дескриптор открытого файла.
  + Если функция возвращает ASYNC\_WORKING, это означает, что операция открытия выполняется. Программа инициализирует переменные N\_C (счётчик), P\_E (флаг) и переходит в состояние 10.
* 10: (\* Открытие файла для чтения, этап 2 \*)
  + OwenFileOpenAsync вызывается повторно для проверки завершения операции открытия.
  + Если res=ASYNC\_DONE, то файл успешно открыт. Проверяется значение handle:
    - Если handle <> 0 — это означает, что файл открыт и программа переходит в состояние 20 (чтение файла).
    - Если handle = 0, это означает, что файл не открыт (возможно, файл не существует или произошла ошибка доступа). Программа переходит в состояние 70 (ошибка чтения файла).
  + Если res < 0 появляется, это означает, что произошла ошибка при открытии файла. Программа переходит в состояние 70.
* 20: (\* Чтение файла, этап 1 \*)
  + Функция OwenFileReadAsync пытается прочитать 4 байта из файла, дескриптор которого хранится в handle, в буфер bufin. Функция является асинхронной.
  + ADR(bufin) - Получает адрес буфера bufin.
  + 4 - Количество байтов для чтения.
  + ADR(result) - Получает адрес переменной result, в которой будет храниться количество фактически прочитанных байтов или код ошибки.
  + Если функция возвращает ASYNC\_WORKING, это означает, что операция чтения выполняется. Программа переходит в состояние 30.
  + Если функция возвращает что-то другое, значит, произошла ошибка при чтении, и программа переходит в состояние 40 (закрытие файла).
* 30: (\* Чтение файла, этап 2 \*)
  + OwenFileReadAsync вызывается повторно для проверки завершения операции чтения.
  + Если res=ASYNC\_DONE, то операция чтения завершена. Проверяется значение result:
    - Если result >= 0, это означает, что данные успешно прочитаны. Программа переходит в состояние 35 (преобразование строки в байт).
    - Если result < 0, это означает, что произошла ошибка при чтении. Программа переходит в состояние 40.
  + Если res < 0, это означает, что произошла ошибка при чтении файла. Программа переходит в состояние 40.
* 35: (\* Преобразование строки в байт и заполнение массива \*)
  + STR1 := RIGHT(bufin, 1); - Извлекает последний символ из буфера bufin и сохраняет его в STR1. Этот символ используется как признак конца записи.
  + STR2 := LEFT(bufin, 3); - Извлекает первые три символа из bufin и сохраняет их в STR2. Предполагается, что эти три символа представляют собой число в строковом виде.
  + MAS[N\_C] := DEC\_TO\_BYTE(STR2); - Преобразует строку STR2 в байт и сохраняет его в массиве MAS под индексом N\_C.
  + IF (STR1 = '\*') THEN ... ELSE ... - Проверяет, является ли последний символ STR1 звездочкой ('\*').
    - Если да, то это означает, что достигнут конец записи. Переменной N\_E присваивается значение N\_C, N\_C сбрасывается в 0, и программа переходит в состояние 40 (закрытие файла).
    - Если нет, это означает, что нужно прочитать следующую запись. N\_C увеличивается на 1, и программа переходит в состояние 20 (чтение файла).
* 40: (\* Закрытие файла, этап 1 \*)
  + Функция OwenFileCloseAsync пытается закрыть файл, дескриптор которого хранится в handle.
  + Если функция возвращает ASYNC\_WORKING, это означает, что операция закрытия выполняется. Программа переходит в состояние 50.
  + Если функция возвращает что-то другое, программа переходит в состояние 60.
* 50: (\* Закрытие файла, этап 2 \*)
  + OwenFileCloseAsync повторно вызывается для проверки завершения операции закрытия.
  + Если res=ASYNC\_DONE, то операция закрытия завершена. Переменной P\_E присваивается значение TRUE. Проверяется значение result:
    - Если result = 0 — это означает, что файл закрыт без ошибок. Программа переходит в состояние 60.
    - Если result <> 0 появляется, это означает, что при закрытии файла произошла ошибка. Программа переходит в состояние 60.
  + Если res < 0 появляется, это означает, что произошла ошибка при закрытии файла. Программа переходит в состояние 60.
* 60: (\*Файл загружен успешно\*)
  + Флаги FL1 и BIT\_3 устанавливаются в значение TRUE, сигнализирует об успешной загрузке файла.
  + Программа переходит в состояние 65 (ожидание).
* 65: (\* Ожидание \*)
  + Программа находится в состоянии ожидания до тех пор, пока не будет выбран другой файл (т. е. пока не будет установлен один из битов BIT\_21 — BIT\_26).
* 70: (\* Ошибка при чтении файла программы \*)
  + Флаг BIT\_0 устанавливается в значение TRUE, что, вероятно, сигнализирует об ошибке при чтении файла.
  + Переменной RG20 присваивается значение 0.
  + Программа переходит в состояние 65 (ожидание).

Переменные:

* state: Целочисленная переменная, определяющая текущее состояние конечного автомата.
* BIT\_21 - BIT\_26: Булевы переменные, соответствующие выбору файла на USB-накопителе.
* NAME: Строковая переменная, содержащая имя файла для чтения.
* RG20: Целочисленная переменная, содержащая номер выбранного файла.
* handle: Целочисленная переменная, содержащая дескриптор открытого файла.
* res: Переменная для хранения результата выполнения асинхронных функций.
* bufin: Строковая переменная (4 символа), используемая как буфер для чтения данных из файла.
* result: Целочисленная переменная, содержащая количество фактически прочитанных байтов или код ошибки.
* MAS: Массив байтов, в который сохраняются данные из файла.
* N\_C: Счетчик прочитанных байтов.
* N\_E: Индекс последнего элемента массива MAS.
* STR1, STR2: Строковые переменные для временного хранения частей строки, прочитанной из файла.
* FL1, BIT\_3, BIT\_0, P\_E: Булевы переменные (флаги), сигнализирующие о различных событиях или состояниях программы.
* TON1: Таймер.
* UT\_1S: Константа времени таймера (1 секунда).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93 | **(\* Загрузка программы \*)**  **IF ((BIT\_21) AND (state = 65)) THEN NAME := 'usb:p1.txt'; RG20 := 1; state := 3; END\_IF**  **IF ((BIT\_22) AND (state = 65)) THEN NAME := 'usb:p2.txt'; RG20 := 2; state := 3; END\_IF**  **IF ((BIT\_23) AND (state = 65)) THEN NAME := 'usb:p3.txt'; RG20 := 3; state := 3; END\_IF**  **IF ((BIT\_24) AND (state = 65)) THEN NAME := 'usb:p4.txt'; RG20 := 4; state := 3; END\_IF**  **IF ((BIT\_25) AND (state = 65)) THEN NAME := 'usb:p5.txt'; RG20 := 5; state := 3; END\_IF**  **IF ((BIT\_26) AND (state = 65)) THEN NAME := 'usb:p6.txt'; RG20 := 6; state := 3; END\_IF**  **CASE state OF**  **3: (\* Пауза \*)**  **TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)**  **IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); FL1 := FALSE; BIT\_3 := FALSE; state := 5; END\_IF**  **5: (\* Открытие файл на чтение, этап 1 \*)**  **res:=OwenFileOpenAsync(NAME, 'r', ADR(handle));**  **IF res = ASYNC\_WORKING THEN**  **N\_C := 0;**  **P\_E := FALSE;**  **state := 10; (\* Нет ошибок, переход на открытие файла, этап 2 \*)**  **END\_IF**  **10:(\* Открытие файла на чтение, этап 2 \*)**  **res:=OwenFileOpenAsync(NAME, 'r', ADR(handle));**  **IF res=ASYNC\_DONE THEN**  **IF handle <> 0 THEN**  **state := 20; (\* Нет ошибок, переход на чтение файла, этап 1 \*)**  **ELSE**  **(\*state := 5; Возврат на открытие файла, этап 1 \*)**  **state := 70;**  **END\_IF**  **ELSIF res < 0 THEN**  **(\*state := 5;\*)**  **state := 70;**  **END\_IF**  **20: (\* Чтение файла, этап 1 \*)**  **res:=OwenFileReadAsync(handle,ADR(bufin),4,ADR(result)); (\* Читаем 4 байта \*)**  **IF res = ASYNC\_WORKING THEN**  **state := 30; (\* Нет ошибок, переход на чтение файла, этап 2\*)**  **ELSE**  **state:=40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)**  **END\_IF**  **30:(\* Чтение файла, этап 2\*)**  **res:=OwenFileReadAsync(handle, ADR(bufin), 4, ADR(result));**  **IF res=ASYNC\_DONE THEN**  **IF result >= 0 THEN (\* Если нет ошибок, и количество считанных данных >=0, то перейти на преобразование строки в байт и заполнение массива \*)**  **state := 35;**  **ELSE**  **state := 40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)**  **END\_IF**  **ELSIF res < 0 THEN**  **state := 40;**  **END\_IF**  **35: (\* Преобразование строки в байт и заполнение массива \*)**  **STR1 := RIGHT(bufin, 1); (\* Выделяем последний символ в строке \*)**  **STR2 := LEFT(bufin, 3); (\* Отсекаем последний символ в строке \*)**  **MAS[N\_C] := DEC\_TO\_BYTE(STR2); (\* Преобразуем строку в байт и помещаем в массив \*)**  **IF (STR1 = '\*') THEN N\_E := N\_C; N\_C := 0; state := 40; ELSE N\_C := N\_C + 1; state := 20; END\_IF**  **40: (\* Закрытие файл, этап 1 \*)**  **res:=OwenFileCloseAsync(handle, ADR(result));**  **IF res=ASYNC\_WORKING THEN**  **state := 50; (\* Нет ошибок, переход на закрытие файла, этап 2 \*)**  **ELSE**  **state := 60; (\* \*)**  **END\_IF**  **50:(\* Закрытие файла, этап 2 \*)**  **res:=OwenFileCloseAsync(handle, ADR(result));**  **IF res=ASYNC\_DONE THEN P\_E := TRUE;**  **IF result = 0 THEN**  **state := 60; (\* \*)**  **ELSE**  **state := 60; (\* \*)**  **END\_IF**  **ELSIF res < 0 THEN**  **state := 60;**  **END\_IF**  **60:(\* Файл загружен успешно \*)**  **FL1 := TRUE; BIT\_3 := TRUE; state := 65;**  **70: (\* Ошибка чтения файла программы \*)**  **BIT\_0 := TRUE; RG20 := 0; state := 65;**  **END\_CASE** |

Листинг 8 – загрузка данных и начало работы.

Режим «Ручное управление»

**Цель:** Разработка логики управления движением каретки по осям X и Y с помощью кнопок на панели оператора ИП320.

**Реализация:**

* **Управление движением:**
  + Использование дискретных входов панели оператора, соответствующие кнопкам управления (вперед, назад, влево, вправо).
  + При нажатии кнопки устанавливается скорость и направление движения для соответствующего двигателя (ШД1 для оси X, ШД2 для оси Y).
  + При отпускании кнопки движение прекращается.
* **Отображение координат:**
  + Считывать текущие значения позиций с энкодеров двигателей.
  + Отображать координаты на панели оператора в режиме реального времени.
* **Обеспечение безопасности:**
  + Установить ограничения на перемещение каретки (максимальные и минимальные значения координат).
  + Контролировать сигналы с датчиков (Д0, Д1, АД0, АД1) и останавливать движение при срабатывании.

На представлен код на ST, реализующий ручное управление каретками станка с ЧПУ. Он использует кнопки на панели управления для перемещения кареток по осям X и Y.

Ключевые компоненты и переменные:

* R9: Переменная состояния, управляющая логикой конечного автомата.
* S\_B\_A\_1: Функциональный блок (вероятно, пользовательский), анализирующий состояние кнопок на панели управления.
  + B0\_C, B1\_C, B2\_C, B3\_C, B4\_C: входы функционального блока, принимающие значения с кнопок.
  + B0, B1, B2, B3, B4: выходы функционального блока, содержащие текущее состояние кнопок (TRUE — нажата, FALSE — не нажата).
* BIT\_32, BIT\_28, BIT\_29, BIT\_30, BIT\_31: Логические переменные, связанные с кнопками на панели управления.
* RG18: Логическая переменная, вероятно, указывающая на режим ручного управления.
* SOMЛогическая переменная, указывающая на то, что ручное управление активно.
* Stop\_Motor(): Функция для остановки моторов.
* TON1: Функциональный блок таймера (TON — таймер с задержкой). Используется для создания паузы.
* Stepper\_3\_Left: Функция управления шаговым двигателем 3 для перемещения влево.
* Stepper\_3\_Right: Функция управления шаговым двигателем 3 для перемещения вправо.
* HD3\_STOPФункция для остановки шагового двигателя 3.
* N1, N2: Переменные, связанные с подсчетом шагов или позиции.
* Stepper\_1: Функция управления шаговым двигателем 1 для перемещения вперед/назад.
  + Task\_Speed: Скорость движения.
  + Accel: Ускорение.
  + Quantity: Количество шагов.
  + Dir: Направление движения (TRUE — вперед, FALSE — назад).
* SteppersConfig\_Pru1MemoryTransfer(STEPPER1\_PRU1\_stepper\_enable := FALSE)Функция отключения шагового двигателя 1.

Описание состояний конечного автомата:

* 0: (\* Ожидание \*)
  + Назначение: ожидание активации режима ручного управления.
  + Действия:
    - S\_B\_A\_1(B0\_C := BIT\_32, B1\_C := BIT\_28, B2\_C := BIT\_29, B3\_C := BIT\_30, B4\_C := BIT\_31 );: Вызов функционального блока для анализа состояния кнопок.
    - IF (S\_B\_A\_1.B0 AND (NOT SOM) AND RG18 = 1) THEN ... END\_IF: Этот условный оператор проверяет:
      * S\_B\_A\_1.B0: Кнопка активации режима ручного управления нажата.
      * (NOT SOM): Ручное управление еще не активировано.
      * RG18 = 1: Режим ручного управления выбран.
    - Если все условия выполнены:
      * Stop\_Motor();: Остановка моторов.
      * BIT\_28 := BIT\_29 := BIT\_30 := BIT\_31 := FALSE;: Сброс состояния кнопок (вероятно, для предотвращения случайных срабатываний).
      * SOM := TRUE;: Установка флага для активации режима ручного управления.
      * R9 := 5;: Переход в состояние 5.
* 5: (\* Пауза \*)
  + Назначение: Небольшая задержка.
  + Действия:
    - TON1(IN := TRUE, PT:= UT\_1S);: Запуск таймера.
    - IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R9 := 10; END\_IF: После завершения таймера переход в состояние 10.
* 10: (\* Ручное управление \*)
  + Назначение: активное управление каретками с помощью кнопок.
  + Действия:
    - IF (S\_B\_A\_1.B1) THEN Stepper\_3\_Left; END\_IF: Если нажата кнопка перемещения влево, вызывается функция Stepper\_3\_Left.
    - IF (S\_B\_A\_1.B2) THEN Stepper\_3\_Right; END\_IF: Если нажата кнопка перемещения вправо, вызывается функция Stepper\_3\_Right.
    - IF ((S\_B\_A\_1.B1 = FALSE) AND (S\_B\_A\_1.B2 = FALSE)) THEN HD3\_STOP; N1 := 0; N2 := 0; END\_IF: Если кнопки перемещения влево и вправо не нажаты, вызывается функция HD3\_STOP для остановки шагового двигателя 3.
    - IF (S\_B\_A\_1.B3) THEN ... END\_IF: Если нажата кнопка перемещения вперед, вызывается функция Stepper\_1 с параметрами для перемещения вперед.
    - IF (S\_B\_A\_1.B4) THEN ... END\_IF: Если нажата кнопка перемещения назад, вызывается функция Stepper\_1 с параметрами для перемещения назад.
    - IF ((S\_B\_A\_1.B3 = FALSE) AND (S\_B\_A\_1.B4 = FALSE)) THEN ... END\_IF: Если кнопки перемещения вперед и назад не нажаты, шаговый двигатель 1 отключается.
    - IF (NOT S\_B\_A\_1.B0) THEN SOM := FALSE; R9 := 0; END\_IF: Если кнопка активации режима ручного управления отпущена, режим ручного управления отключается, и автомат переходит в состояние 0 (ожидание).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **(\* Ручное управление каретками \*)**  **S\_B\_A\_1(B0\_C := BIT\_32, B1\_C := BIT\_28, B2\_C := BIT\_29, B3\_C := BIT\_30, B4\_C := BIT\_31 ); (\* Анализ состояния кнопок \*)**  **CASE R9 OF**  **0: (\* Ожидание \*)**  **IF (S\_B\_A\_1.B0 AND (NOT SOM) AND RG18 =1) THEN**  **Stop\_Motor();**  **BIT\_28 := BIT\_29 := BIT\_30 := BIT\_31 := FALSE;**  **SOM := TRUE;**  **R9 := 5;**  **END\_IF**  **5: (\* Пауза \*)**  **TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)**  **IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R9 := 10; END\_IF**  **10: (\* Ручное управление \*)**  **(\*S\_B\_A\_1(B0\_C := BIT\_32, B1\_C := BIT\_28, B2\_C := BIT\_29, B3\_C := BIT\_30, B4\_C := BIT\_31 ); Анализ состояния кнопок \*)**  **IF (S\_B\_A\_1.B1) THEN Stepper\_3\_Left; END\_IF (\* Влево \*)**  **IF (S\_B\_A\_1.B2) THEN Stepper\_3\_Right; END\_IF (\* Вправо \*)**  **IF ((S\_B\_A\_1.B1 = FALSE) AND (S\_B\_A\_1.B2 = FALSE)) THEN HD3\_STOP; N1 := 0; N2 := 0; END\_IF (\* Стоп \*)**  **IF (S\_B\_A\_1.B3) THEN (\* Вперед \*)**  **Stepper\_1(Task\_Speed := 5000, Accel := 50000, Quantity := 50000, Dir := TRUE);**  **END\_IF**  **IF (S\_B\_A\_1.B4) THEN (\* Назад \*)**  **Stepper\_1(Task\_Speed := 5000, Accel := 50000, Quantity := 50000, Dir := FALSE);**  **END\_IF**  **IF ((S\_B\_A\_1.B3 = FALSE) AND (S\_B\_A\_1.B4 = FALSE)) THEN (\* Стоп \*)**  **SteppersConfig\_Pru1MemoryTransfer(STEPPER1\_PRU1\_stepper\_enable := FALSE);**  **END\_IF**  **IF (NOT S\_B\_A\_1.B0) THEN SOM := FALSE; R9 := 0; END\_IF**  **END\_CASE** |

Листинг 9 – ручное управление, с панели оператора.

## Интеграция и отладка программного обеспечения

В этом разделе мы опишем процесс интеграции разработанных функциональных модулей в единую систему управления станком КТФ-8, а также методы отладки, тестирования и оптимизации программного обеспечения.

Интеграция разработанных функциональных модулей в единую систему управления.

На этом этапе необходимо объединить отдельные функциональные модули (панель оператора, режимы обработки, нулевое положение, фрезерование, загрузка, ручное управление) в единую программу, которая будет управлять станком КТФ-8.

#### Создание основной программы

В CODESYS создайте главную программу, которая будет координировать работу всех функциональных модулей.

Главная программа должна содержать логику инициализации всех модулей, обработки событий (например, нажатия кнопок на панели оператора), вызова соответствующих функциональных блоков и обработки ошибок.

#### Организация кода:

* Использование графического языка (SFC) для представления полного процесса программы. Это обеспечит модульность, повторное использование кода и упростит отладку.

#### Пример структуры главной программы ():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | **PROGRAM PLC\_PRG**  **(\***  **Шаг винта по оси Х - 5 мм/об**  **Шаг винта по оси Y - 4 мм/об**  **Число импульсов на 1 оборот энкодера - 400**  **\*)**  **VAR**  **PEREXOD: BOOL:=FALSE;**  **handle:DWORD:=0;**  **res: ASYNC\_RET\_VALUE; (\* То, что возвращает функция \*)**  **bufin: STRING(4) := ' '; (\* Строка, в которую читается файл \*)**  **STR1:STRING(1) := ' '; (\* Промежуточная строка \*)**  **STR2:STRING(3) := ' '; (\* Промежуточная строка \*)**  **NAME: STRING(10) := 'usb:p1.txt'; (\* Имя файла программы \*)**  **NAME1: STRING(13) := 'usb:param.txt'; (\* Имя файла параметров фрезерования \*)**  **result:DWORD;**  **FL1: BOOL; (\* Признак окончания загрузки файла программы \*)**  **FL2: BOOL; (\* Признак окончания загрузки параметров фрезерования \*)**  **L100:STRING(240); (\* Строка, в которую читают файл \*)**  **L1:STRING(16); (\* Промежуточная строка \*)**  **L2:STRING(16); (\* Промежуточная строка \*)**  **L3:STRING(16); (\* Промежуточная строка \*)**  **L4:STRING(16); (\* Промежуточная строка \*)**  **L5:STRING(16); (\* Промежуточная строка \*)**  **L6:STRING(16); (\* Промежуточная строка \*)**  **L7:STRING(16); (\* Промежуточная строка \*)**  **L8:STRING(16); (\* Промежуточная строка \*)**  **L9:STRING(16); (\* Промежуточная строка \*)**  **L10:STRING(16); (\* Промежуточная строка \*)**  **L11:STRING(16); (\* Промежуточная строка \*)**  **L12:STRING(16); (\* Промежуточная строка \*)**  **L13:STRING(16); (\* Промежуточная строка \*)**  **L14:STRING(16); (\* Промежуточная строка \*)**  **L15:STRING(16); (\* Промежуточная строка \*)**  **L32:STRING(32);**  **L48:STRING(48);**  **L64:STRING(64);**  **L80:STRING(80);**  **L96:STRING(96);**  **L112:STRING(112);**  **L128:STRING(128);**  **L144:STRING(144);**  **L160:STRING(160);**  **L176:STRING(176);**  **L192:STRING(192);**  **L208:STRING(208);**  **L224:STRING(224);**  **L240:STRING(240);**  **S\_B\_A\_1: S\_B\_A;**  **Embedment: DWORD; (\* Заглубление, число импульсов \*)**  **Accel\_1: DWORD; (\* Ускорение по Y \*)**  **Accel\_m: DWORD := 16000000; (\* Максимальное ускорение по Y \*)**  **Dir\_1: BOOL; (\* Направление движения по оси Y \*)**  **Quantity\_1: DWORD; (\* Количество импульсов по оси Y для ШД1 \*)**  **Quantity\_3: DWORD; (\* Количество импульсов по оси X для ШД3 \*)**  **Step\_TIME: REAL := 0.006; (\* Время отработк шага по оси Х, сек \*)**  **Step\_TIME1: REAL := 0.006; (\* Увеличенное время отработк шага по оси Х, сек \*)**  **P\_D1: REAL := 0.01; (\* Цена дискреты, 1 в массиве MAS[] означает перемещение на 0,01 мм по оси Y \*)**  **(\*P\_D3: REAL := 0.1; (\* Цена дискреты, 0,1 мм по оси X \*)\*)**  **P\_D3: REAL := 2.0; (\* Цена дискреты, 2.0 мм по оси X \*)**  **NUMBER\_I1: WORD := 200; (\* Число импульсов на 1 оборот вала ШД1 \*)**  **NUMBER\_I3: WORD := 200; (\* Число импульсов на 1 оборот вала ШД3 \*)**  **TON2: TON ;**  **PP\_X: REAL := 5.0; (\* Шаг винта по оси Х - 5 мм/об \*)**  **FL\_STEP1: BOOL; (\* Флаг отработки перемещения по оси Y \*)**  **FL\_STEP3: BOOL; (\* Флаг отработки перемещения по оси X \*)**  **COUN: DWORD; (\* Число импульсов, отработанное ШД3 \*)**  **END\_VAR** |

Листинг 10 – основная программа, выполняемая циклически (PRG-SFC).

#### Управление потоком выполнения:

* Использование конечные автоматы для управления потоком выполнения программы. Это позволит четко определить состояния системы и переходы между ними.
* Определение состояния для каждого режима работы, ошибки, ожидания и т. д.
* Использование события и условия для переключения между состояниями.

Отладка и тестирование программы на физическом оборудовании (станке КТФ-8) для выявления и устранения ошибок

На этом этапе необходимо тщательно протестировать программу на реальном станке КТФ-8 для выявления и устранения ошибок.

#### Подготовка к отладке:

* Проверка на предмет верного соединения между ПЛК и станком КТФ-8.
* Проверка настроек CODESYS для связи с ПЛК (IP-адрес, COM-порт и т. д.).
* Загрузка программы в ПЛК.
* Подготовка контрольных примеров.

#### Методы отладки:

* **Режим онлайн-мониторинга CODESYS (Online Monitoring):** позволяет просматривать значения переменных, состояние функциональных блоков и ход выполнения программы в режиме реального времени.
* **Точки останова (Breakpoints):** позволяют приостановить выполнение программы в заданных точках для анализа состояния переменных и хода выполнения.
* **Пошаговое выполнение (Step-by-step Execution):** позволяет выполнять программу по одной строке кода, отслеживая изменение значений переменных.
* **Использование симулятора CODESYS (SoftPLC):** позволяет тестировать программу на компьютере без подключения к реальному оборудованию. Это полезно для выявления логических ошибок и проверки основных функций.

#### Стратегия тестирования:

* **Интеграционное тестирование:** проверка взаимодействия между модулями, чтобы убедиться, что данные передаются правильно и модули работают согласованно.
* **Системное тестирование:** проверка всей системы в целом, чтобы убедиться, что она соответствует требованиям технического задания.
* **Регрессионное тестирование:** после внесения изменений в код повторите тесты, чтобы убедиться, что новые изменения не нарушили работу существующих функций.

#### Устранение ошибок:

* Использование отладчика CODESYS для пошагового выполнения кода и анализа значений переменных.
* Проверка логических условий, вычислений и параметров настройки.
* Используйте систему контроля версий для отслеживания изменений в коде и возможности возврата к предыдущим версиям.

Проведение испытаний для подтверждения соответствия программы требованиям технического задания

На этом этапе необходимо провести формальные испытания, чтобы убедиться, что программа соответствует всем требованиям технического задания.

#### Разработка плана испытаний:

* Определить все требования технического задания, которые необходимо проверить.
* Для каждого требования разработать тестовый пример с четким описанием входных данных, ожидаемых результатов и критериев успеха.
* Определите порядок проведения испытаний и ответственных лиц.

#### Проведение испытаний:

* Выполнение тестовых примеров на физическом оборудовании.

#### Анализ результатов испытаний:

* Сравнение фактических результатов с ожидаемыми.
* Выявление несоответствия и определение причины.
* Устранение ошибки в коде и повторение испытания.

## Обеспечение надежности и безопасности

В данном разделе подробно описывается реализация механизмов защиты от некорректных действий пользователя, аварийной остановки и обеспечения надежной работы программы при сбоях электропитания для системы управления станком КТФ-8 в CODESYS.

Реализация механизма защиты от некорректных действий пользователя.

Цель данного механизма - предотвратить возникновение ошибок и повреждений оборудования в результате неправильных действий оператора, таких как ввод некорректных данных или запуск несовместимых режимов работы.

#### Проверка входных данных:

* **Диапазон значений:** Для каждого поля ввода параметров необходимо установить минимальное и максимальное допустимые значения. При вводе значения, выходящего за пределы диапазона, необходимо отобразить сообщение об ошибке и отклонить ввод.
  + Реализация: Использовать функции сравнения и условные операторы в ST для проверки значения.
* **Тип данных:** Убедитесь, что пользователь вводит данные правильного типа (число, текст, дата и т.д.).
  + Реализация: Использовать функции преобразования типов (STRING\_TO\_REAL, STRING\_TO\_INT и т.д.) и проверять успешность преобразования.
* **Формат данных:** Для полей ввода, требующих определенного формата (например, дата, время, координаты), необходимо проверять соответствие введенных данных заданному формату.
  + Реализация: Использовать функции работы со строками для проверки формата.
* **Совместимость режимов работы:** Необходимо реализовать логику, которая предотвращает запуск несовместимых режимов работы. Например, нельзя запускать режим фрезерования, если не установлен фрезерный инструмент.
  + Реализация: Использовать флаги и логические условия для проверки возможности запуска режима.

#### Обработка ошибок:

* **Общие ошибки:** Разработать механизм обработки общих ошибок, таких как ошибки связи с датчиками, ошибки при работе с файлами, ошибки при выполнении вычислений.
  + Реализация: Использовать блоки обработки ошибок (TRY…CATCH) и проверять коды возврата функций.
* **Реакция на ошибки:** При возникновении ошибки необходимо:
  + Остановить выполнение программы (если это необходимо).
  + Вывести сообщение об ошибке на панель оператора.
  + Перевести систему в безопасное состояние.

Разработка алгоритма аварийной остановки всех двигателей при нажатии кнопки «СТОП» или срабатывании аварийных датчиков.

Цель данного алгоритма - немедленно остановить все движущиеся части станка при возникновении аварийной ситуации, чтобы предотвратить повреждение оборудования и травмы оператора.

#### Кнопка “СТОП” и аварийные датчики:

* Использование дискретного входа ПЛК, подключенного к кнопке “СТОП”.
* Подключение аварийных датчиков (выхода за пределы рабочей зоны и т.д.) к дискретным входам ПЛК.
* При нажатии кнопки “СТОП” или срабатывании аварийного датчика установить соответствующий флаг.

#### Реализация алгоритма аварийной остановки:

* Создать отдельный функциональный блок, который будет отвечать за аварийную остановку.
* Внутри этого блока реализовать следующие действия:
  + Остановить все двигатели.
  + Перевести систему в безопасное состояние.
* Вызывать функциональный блок при установке флага аварийной остановки.

Код программы, предложенный в , реализует аварийную остановку станка с ЧПУ.

Ключевые компоненты и переменные:

* BIT\_33 Логическая переменная, связанная с кнопкой или переключателем аварийного останова на панели управления.
* Stop\_Motor(): Функция для немедленной остановки всех двигателей станка.
* S\_B\_A\_1: Функциональный блок, анализирующий состояние кнопок на панели управления. Здесь используется для установки определенных значений, связанных с управлением.
  + B0\_C := TRUE: Установка значения, которое, связано с активацией автоматического режима
  + B1\_C := FALSE, B2\_C := FALSE, B3\_C := FALSE, B4\_C := FALSE: Сброс состояния кнопок управления движением.
* RG18: Переменная, указывающая текущий режим работы станка (0 — автоматический, 1 — ручной). В данном коде устанавливается в 0, что соответствует переходу в автоматический режим (или блокировке ручного режима).
* BIT\_10 Логическая переменная, используемая для индикации процесса поиска нулевого положения. Здесь сбрасывается в FALSE, чтобы выключить индикатор в окне возврата в «0».
* Reset\_R Функция, предназначенная для сброса каких-либо внутренних переменных или состояний системы, связанных с процессом обработки или управления станком.
* SOM: Флаг, указывающий на то, что станок выполняет какую-либо операцию.

Описание работы кода:

1. IF (BIT\_33) THEN ... END\_IF Этот условный оператор проверяет, нажата ли кнопка аварийного останова (BIT\_33 = TRUE). Если кнопка нажата, выполняется следующий блок кода:
   * Stop\_Motor();: Функция немедленной остановки всех двигателей станка. Это критически важно для предотвращения дальнейшего движения и потенциального повреждения оборудования или травм.
   * S\_B\_A\_1(B0\_C := TRUE, B1\_C := FALSE, B2\_C := FALSE, B3\_C := FALSE, B4\_C := FALSE );: Установка состояния кнопок через функциональный блок S\_B\_A\_1. Значение TRUE для B0\_C может перевести систему в автоматический режим. Сброс остальных кнопок предотвращает неконтролируемое перемещение кареток при снятии аварийной блокировки.
   * RG18 := 0;: Принудительный переход в автоматический режим (или блокировка ручного управления). Это может быть сделано для предотвращения случайного запуска в ручном режиме после аварийного останова.
   * BIT\_10 := FALSE;: Выключение индикации процесса поиска нулевого положения (если он был активен). Это необходимо, чтобы не вводить пользователя в заблуждение после аварийного останова.
   * Reset\_R: Сброс внутренних переменных или состояний системы (каких именно — нужно смотреть в реализации функции). Это важно для обеспечения корректного запуска после устранения аварийной ситуации.
   * SOM := FALSE;: сброс флага выполнения какой-либо операции.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **(\* Аварийный останов \*)**  **IF (BIT\_33) THEN**  **(\*RG1 := 0;\*)**  **Stop\_Motor();**  **S\_B\_A\_1(B0\_C := TRUE, B1\_C := FALSE, B2\_C := FALSE, B3\_C := FALSE, B4\_C := FALSE );**  **RG18 := 0; (\* Режим "Автоматический" \*)**  **BIT\_10 := FALSE; (\* Гашение индикатора в окне Возврат в "0" \*)**  **Reset\_R;**  **SOM := FALSE;**  **END\_IF** |

Листинг 11 – программа аварийной остановки.

#### Приоритет аварийной остановки:

* Алгоритм аварийной остановки должен иметь наивысший приоритет и выполняться немедленно при возникновении аварийной ситуации.
* Не должно быть возможности игнорировать или отключить аварийную остановку.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана и реализована система программного управления для токарно-фрезерного станка КТФ-8, соответствующая требованиям, сформулированным в техническом задании. В процессе работы были успешно решены поставленные задачи, что позволило создать функциональное и надежное программное обеспечение, способное выполнять широкий спектр операций по обработке деталей.

В рамках первого этапа – разработки архитектуры программного обеспечения – была определена модульная структура программы, позволяющая эффективно организовать и разделить различные функциональные блоки. Разработаны алгоритмы работы для каждого модуля, спроектирована структура данных для хранения и обработки информации, а также произведен выбор оптимальных языков программирования (ST) и библиотек CODESYS, обеспечивающих высокую производительность и гибкость системы.

На втором этапе – реализации функциональных модулей – были разработаны и протестированы все ключевые компоненты системы: панель оператора с интуитивно понятным графическим интерфейсом, режимы обработки (включая «1 проход», «2 прохода» и режимы с фрезерованием), режим “Нулевое положение”, режим “Фрезерование”, режим “Загрузка” и режим “Ручное управление”. Особое внимание было уделено управлению шаговыми двигателями [5], контролю датчиков и обеспечению безопасной и эффективной работы станка.

Этап интеграции и отладки включал объединение разработанных модулей в единую систему и интенсивное тестирование на физическом оборудовании (станке КТФ-8). В результате выявленные и устранены ошибки, что позволило убедиться в соответствии разработанного программного обеспечения требованиям технического задания и обеспечить его стабильную и надежную работу.

На заключительном этапе были реализованы механизмы обеспечения надежности и безопасности системы, включая защиту от некорректных действий пользователя и алгоритм аварийной остановки всех двигателей. Эти меры направлены на предотвращение возможных аварийных ситуаций и защиту оборудования и оператора.

В результате выполнения данной ВКР было разработано комплексное программное обеспечение, которое:

* Автоматизирует процесс управления токарно-фрезерным станком КТФ-8.
* Повышает точность и эффективность обработки деталей.
* Обеспечивает безопасность работы станка.
* Предоставляет оператору удобный и интуитивно понятный интерфейс управления.

Перспективы дальнейшего развития работы:

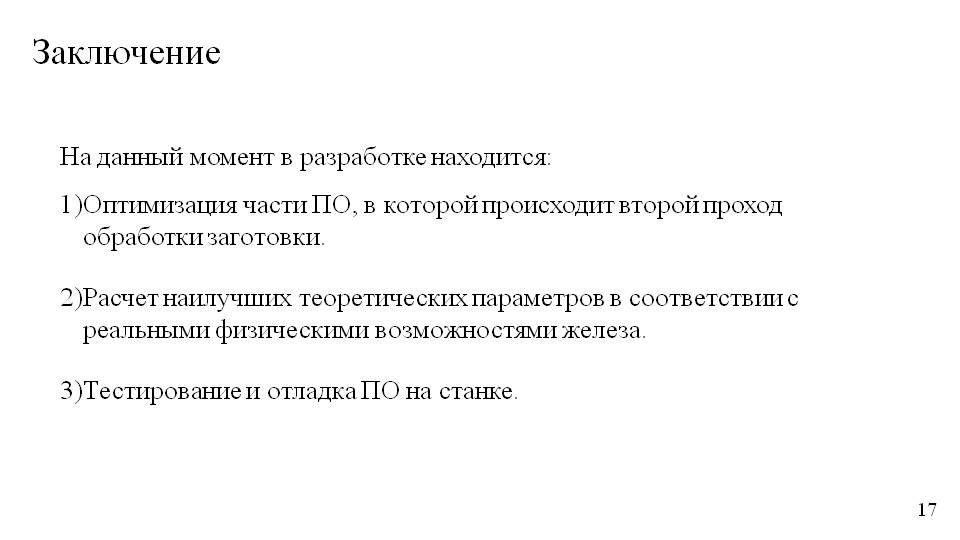
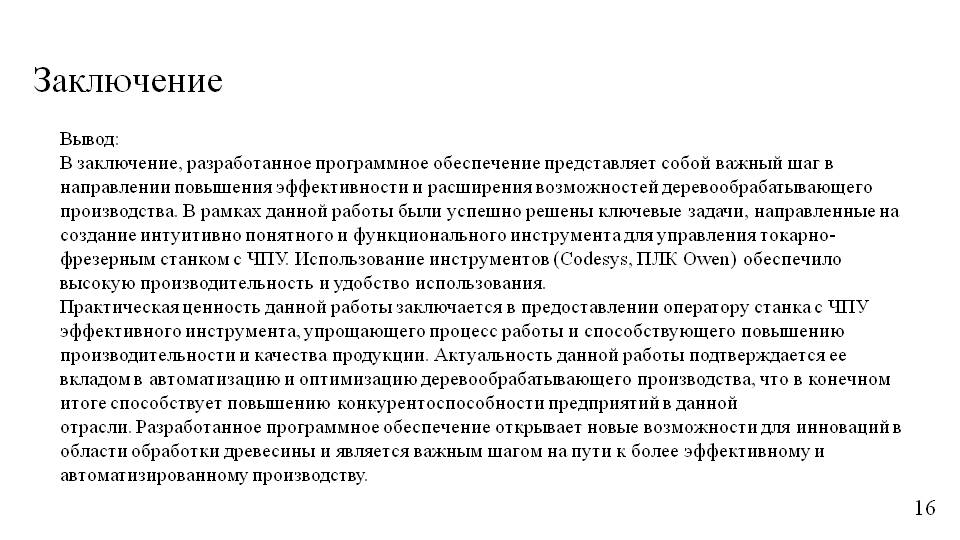
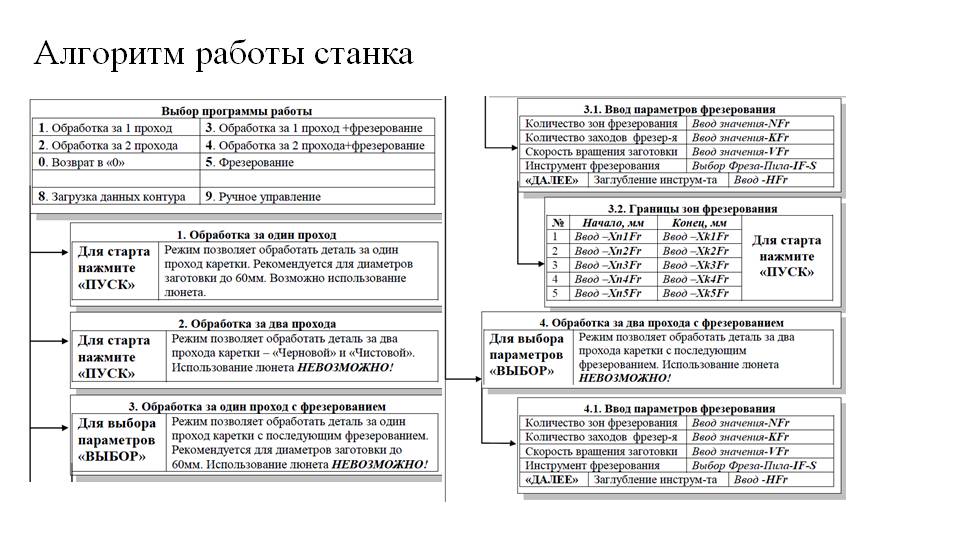
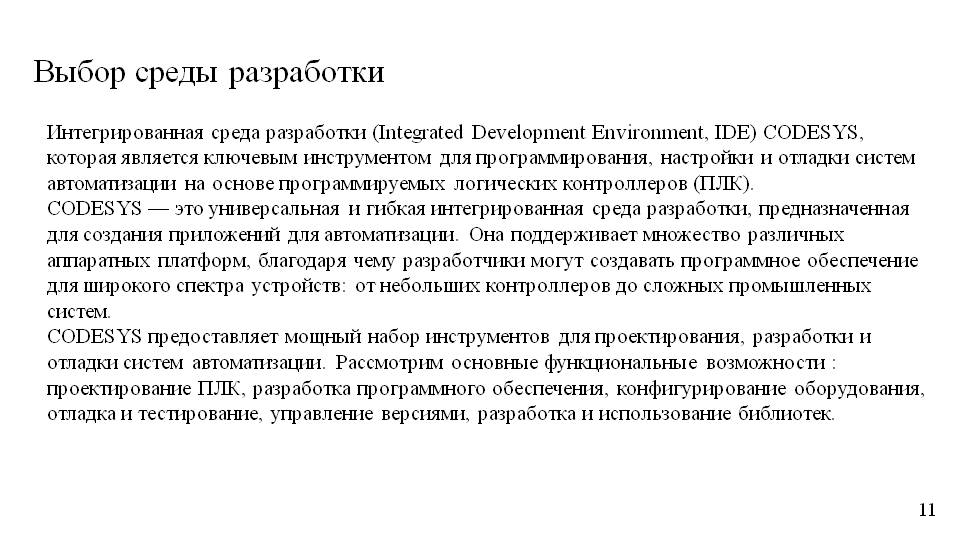
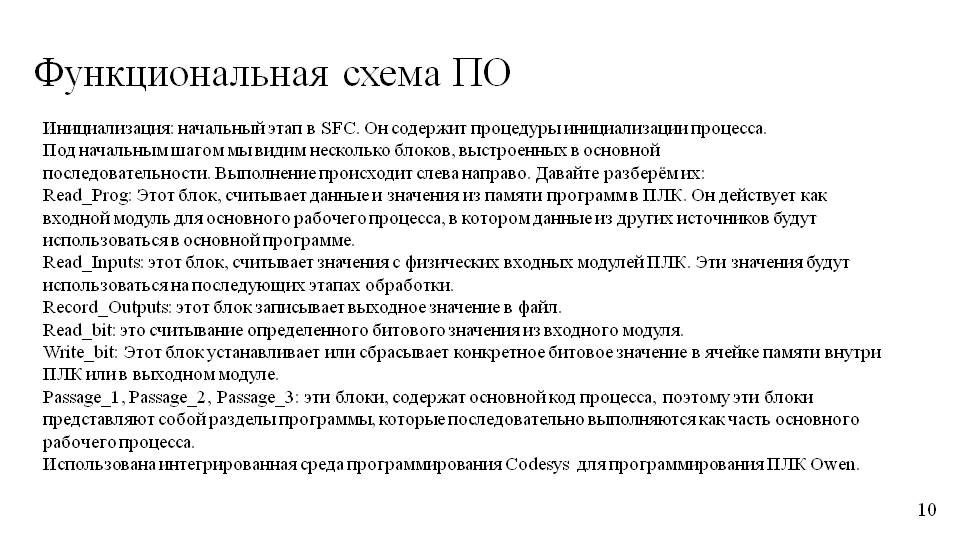
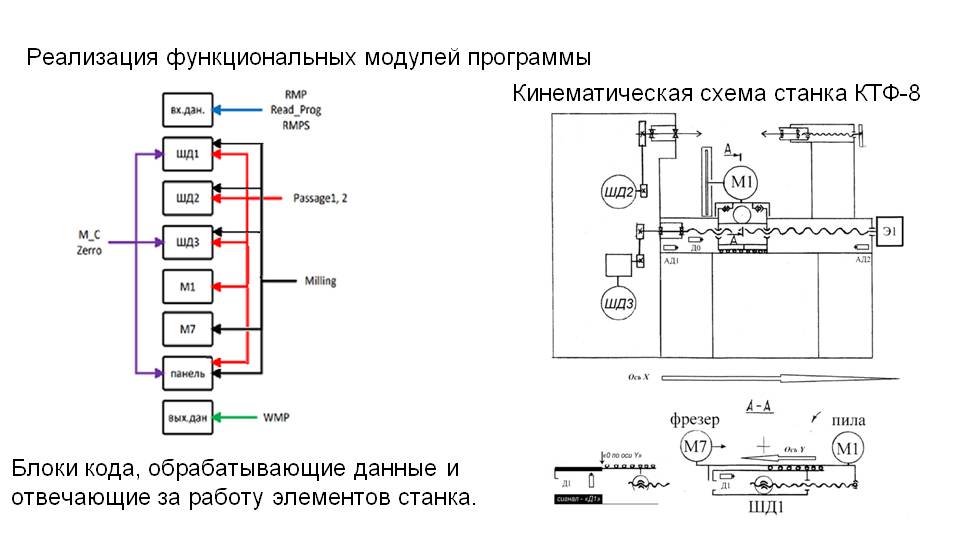
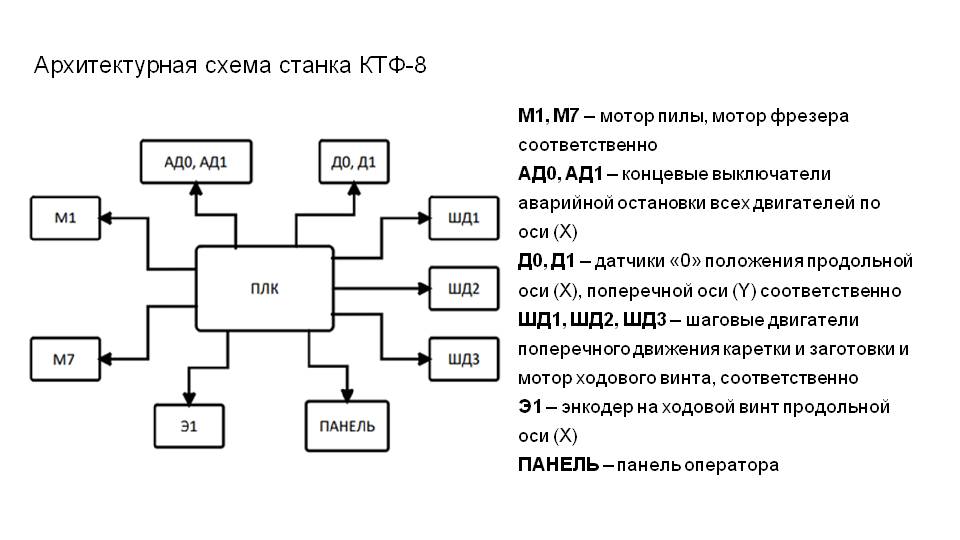
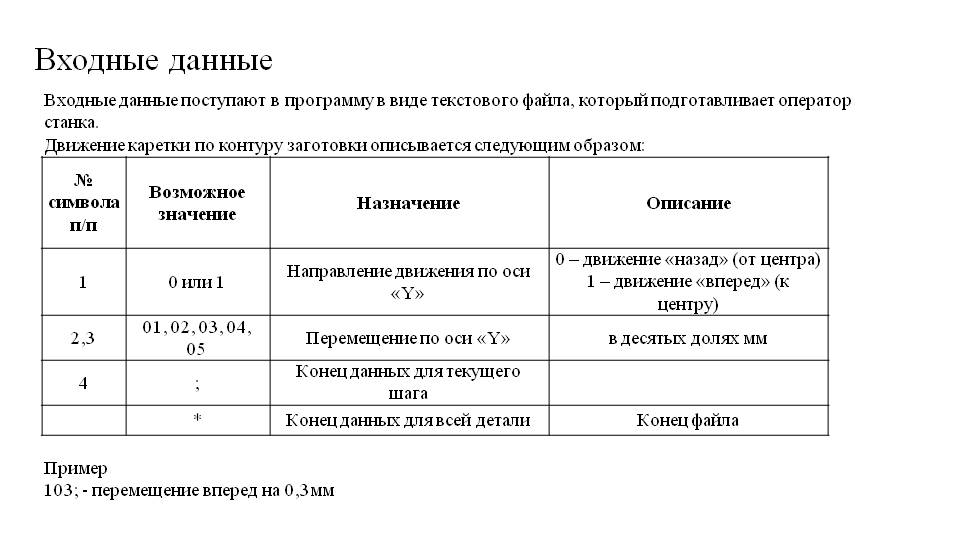
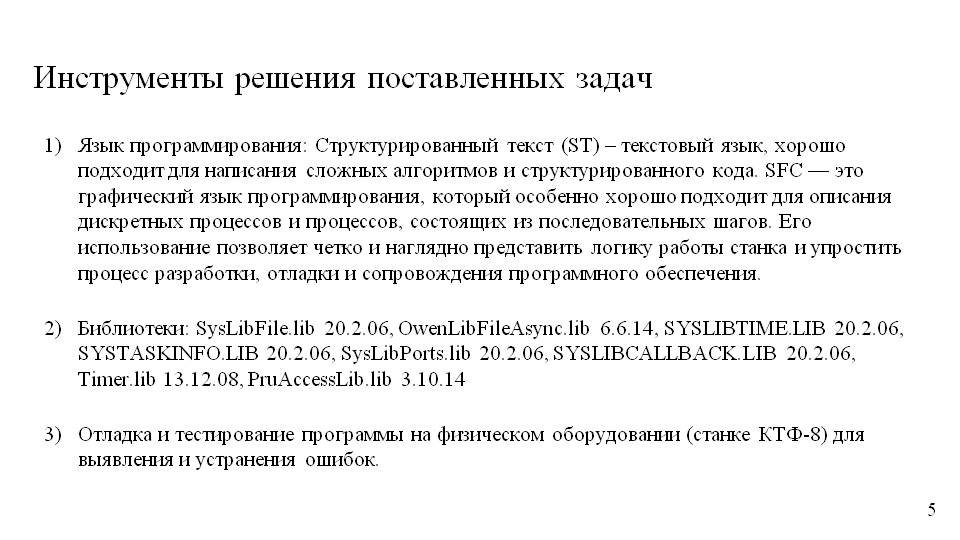
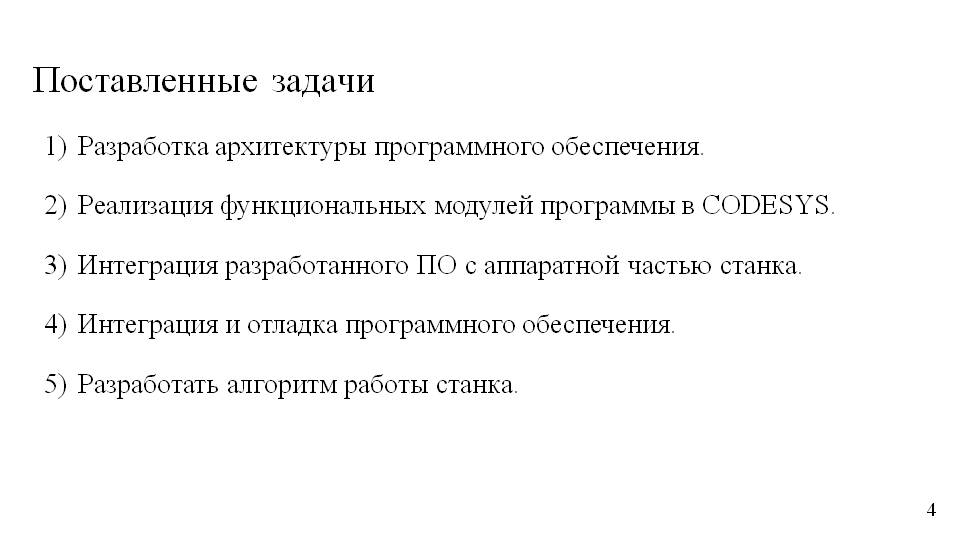
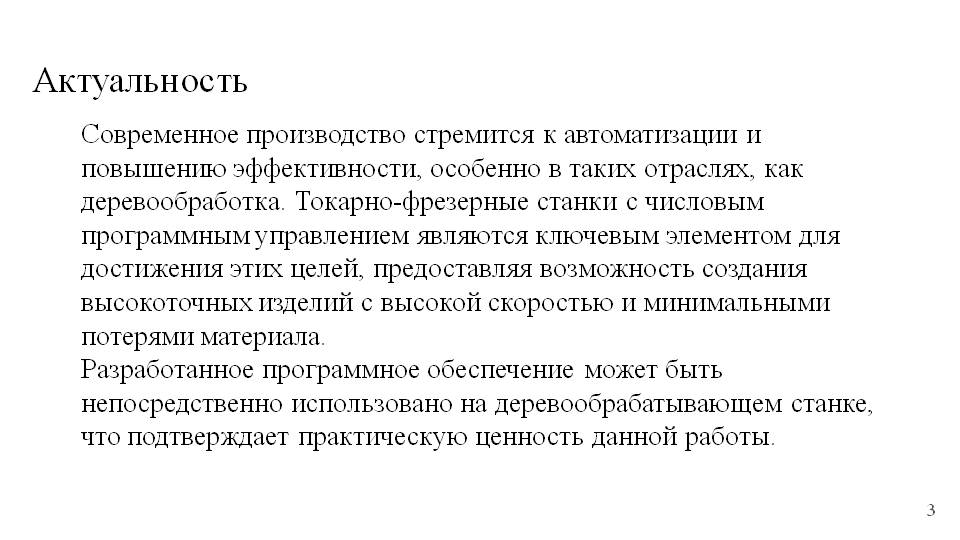
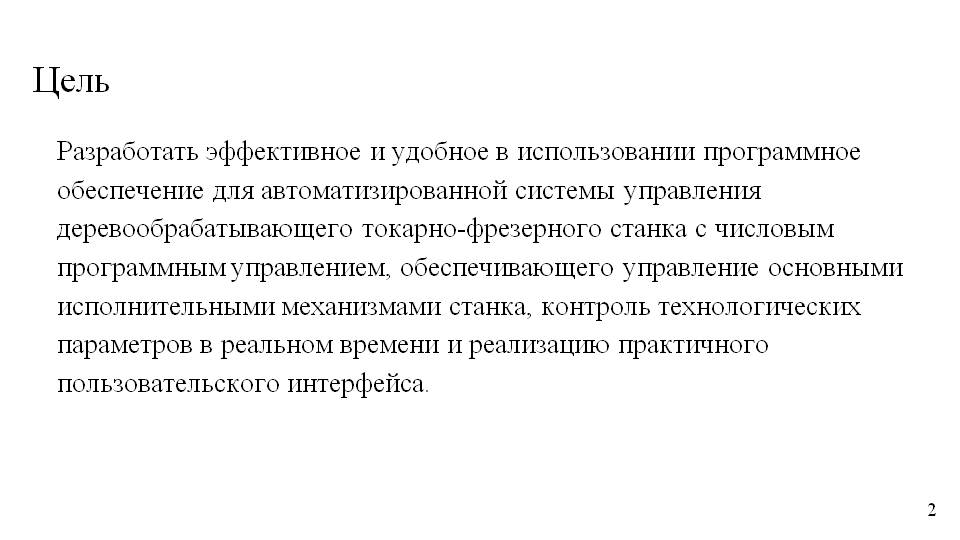
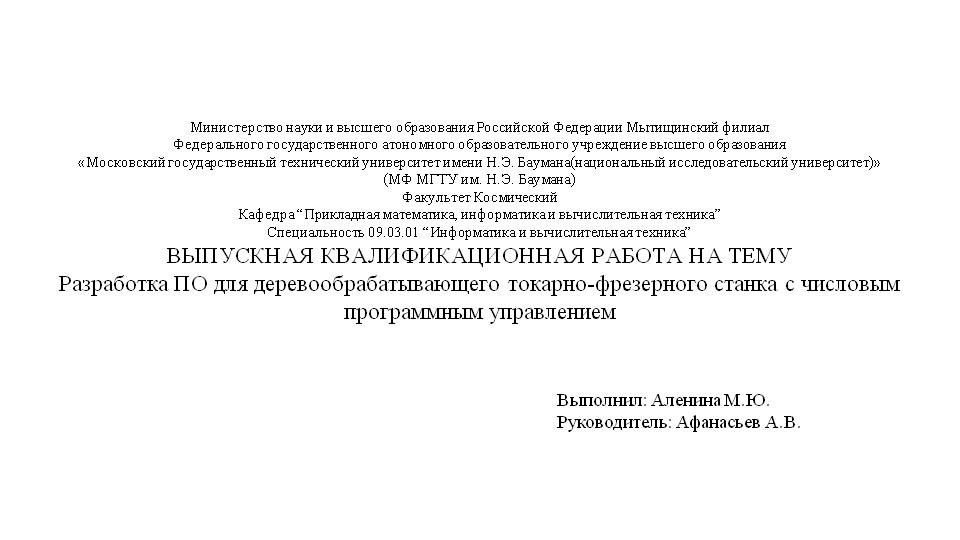
* Реализация системы автоматической загрузки и выгрузки деталей.
* Разработка алгоритмов оптимизации траектории инструмента для повышения производительности и качества обработки.
* Внедрение системы удаленного мониторинга и диагностики состояния станка.
* Расширение функциональности панели оператора и добавление новых функций для управления станком.

Данная выпускная квалификационная работа демонстрирует практическое применение полученных теоретических знаний и навыков в области программирования ПЛК и разработки систем автоматизации. Результаты работы могут быть использованы для модернизации и повышения эффективности работы токарно-фрезерных станков КТФ-8 и подобных им.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Панель оператора ИП320. Руководство по эксплуатации КУВФ.421449.002 РЭ / [разработчик не указан]. - [Москва] : [Издательство не указано], 2023. - 50 с. - Код ОКП 40 3230.
2. ПЛК 110 Программируемый логический контроллер. Руководство по эксплуатации / [Компания ОВЕН]. - Москва : ОВЕН, 2020. - 100 с. - Рег. № 1738.
3. Программирование программируемых логических контроллеров ОВЕН ПЛК110 и ПЛК160. Руководство пользователя. Версия 1.9. - Москва, 2010. - 150 с.
4. Драйвер PLD86 для шаговых двигателей. [Электронный ресурс] // PureLogic.ru. URL: <https://purelogic.ru/docs/elektronika/driver_stepmotor_pld86_info.pdf> (дата обращения: 12.11.2024).
5. PLD8220 G3. Руководство пользователя. [Электронный ресурс] // PureLogic.ru. URL: <https://purelogic.ru/docs/elektronika/driver_stepmotor_pld8220_g3_user_manual_ru.pdf> (дата обращения: 15.11.2024).
6. Библиотеки CODESYS V2. [Электронный ресурс] // ОВЕН. URL: <https://owen.ru/product/codesys_v2/libraries> (дата обращения: 14.12.2024).
7. Руководство пользователя CoDeSys V2.3. [Электронный ресурс] // ОВЕН. URL: <https://owen.ru/uploads/134/codesys_v23_ru.pdf> (дата обращения: 10.12.2024).
8. Фрезерный станок с ЧПУ DML 600. [Электронный ресурс] // Stanki.ru. URL: <https://www.stanki.ru/video/equipment/videos/8954248_223823167456239430/> (дата обращения: 25.03.2025).
9. Автоматизированный электропривод: учебник для вузов / И. И. Петренко [и др.] ; под ред. И. И. Петренко. – 2-е изд., перераб. и доп. – Москва : Юрайт, 2023. – 447 с. – (Высшее образование). – ISBN 978-5-534-05624-1.
10. Кирьянов, В. В. ЧПУ. Теория и практика : учебное пособие для вузов / В. В. Кирьянов. – 2-е изд., испр. и доп. – Москва : Юрайт, 2023. – 428 с. – (Высшее образование). – ISBN 978-5-534-10948-0.
11. Системы числового программного управления: учебник для вузов / [В. О. Антонов и др.] ; под редакцией Ю. М. Соломенцева. – Москва : Машиностроение, 2005. – 592 с. – ISBN 5-217-03244-1.
12. Электропривод и автоматизация типовых производственных установок : учебник для вузов / В. И. Ключев [и др.] ; под ред. М. М. Соколова. – 4-е изд., стер. – Москва : Академия, 2013. – 352 с. – (Высшее профессиональное образование. Машиностроение). – ISBN 978-5-7695-9844-3.

# ПРИЛОЖЕНИЕ А

****

# ПРИЛОЖЕНИЕ Б

УТВЕРЖДАЮ

Зав. кафедрой К-3 МФ

А.В.Корольков.

« » 2025 г.

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Спецификация

ЛИСТ УТВЕРЖДЕНИЯ

643.МГУЛ.090301-45-ЛУ

Студент

М.Ю.Аленина

« » 2025 г.

Руководитель

А.В. Афанасьев

« » 2025 г.

Нормоконтролер

А.В. Чернышов

« » 2025 г.

УТВЕРЖДЕН 643.МГУЛ.090301-45-ЛУ

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Спецификация 643.МГУЛ.090301-45

Листов 3

|  |  |  |
| --- | --- | --- |
| Обозначение | Наименование | Примечание |
|  | Документация |  |
|  |  |  |
| 643.МГУЛ.090301-45-ЛУ | Программное обеспечение | Размножать |
|  | для | по |
|  | деревообрабатывающего | указанию |
|  | токарно-фрезерного |  |
|  | станка с числовым |  |
|  | программным управлением |  |
|  | спецификация |  |
|  | Лист утверждения |  |
| 643.МГУЛ.090301-45 12 01 | Программное обеспечение |  |
|  | для |  |
|  | деревообрабатывающего |  |
|  | токарно-фрезерного |  |
|  | станка с числовым |  |
|  | программным управлением |  |
|  | Текст программы |  |
| 643.МГУЛ.090301-45 12 01-ЛУ | Программное обеспечение | Размножать |
|  | для | по |
|  | деревообрабатывающего | указанию |
|  | токарно-фрезерного |  |
|  | станка с числовым |  |
|  | программным управлением |  |
|  | Текст программы |  |
|  | Лист утверждения |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Обозначение | Наименование | Примечание |
|  |  |  |
| 643.МГУЛ.090301-45 34 01 | Программное обеспечение |  |
|  | для |  |
|  | деревообрабатывающего |  |
|  | токарно-фрезерного |  |
|  | станка с числовым |  |
|  | программным управлением |  |
|  | Руководство оператора |  |
|  |  |  |
| 643.МГУЛ. 090301-45 34 01-ЛУ | Программное обеспечение | Размножать |
|  | для | по |
|  | деревообрабатывающего | указанию |
|  | токарно-фрезерного |  |
|  | станка с числовым |  |
|  | программным управлением |  |
|  | Руководство оператора |  |
|  | Лист утверждения |  |
|  |  |  |
|  |  |  |
|  |  |  |

УТВЕРЖДАЮ

Зав. кафедрой К-3 МФ

А.В.Корольков.

« » 2025 г.

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

643.МГУЛ.090301-45 12 01-ЛУ

Студент

М.Ю.Аленина

« » 2025 г.

Руководитель

А.В. Афанасьев

« » 2025 г.

Нормоконтролер

А.В. Чернышов

« » 2025 г.

УТВЕРЖДЕН 643.МГУЛ.090301-45 12 01-ЛУ

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Текст программы 643.МГУЛ.090301-45 12 01

Листов 30

### Файл PLC\_PRG

PROGRAM PLC\_PRG

(\*

Шаг винта по оси Х - 5 мм/об

Шаг винта по оси Y - 4 мм/об

Число импульсов на 1 оборот энкодера - 400

\*)

VAR

PEREXOD: BOOL:=FALSE;

handle:DWORD:=0;

res: ASYNC\_RET\_VALUE; (\* То, что возвращает функция \*)

bufin: STRING(4) := ' '; (\* Строка, в которую читается файл \*)

STR1:STRING(1) := ' '; (\* Промежуточная строка \*)

STR2:STRING(3) := ' '; (\* Промежуточная строка \*)

NAME: STRING(10) := 'usb:p1.txt'; (\* Имя файла программы \*)

NAME1: STRING(13) := 'usb:param.txt'; (\* Имя файла параметров фрезерования \*)

result:DWORD;

FL1: BOOL; (\* Признак окончания загрузки файла программы \*)

FL2: BOOL; (\* Признак окончания загрузки параметров фрезерования \*)

L100:STRING(240); (\* Строка, в которую читают файл \*)

L1:STRING(16); (\* Промежуточная строка \*)

L2:STRING(16); (\* Промежуточная строка \*)

L3:STRING(16); (\* Промежуточная строка \*)

L4:STRING(16); (\* Промежуточная строка \*)

L5:STRING(16); (\* Промежуточная строка \*)

L6:STRING(16); (\* Промежуточная строка \*)

L7:STRING(16); (\* Промежуточная строка \*)

L8:STRING(16); (\* Промежуточная строка \*)

L9:STRING(16); (\* Промежуточная строка \*)

L10:STRING(16); (\* Промежуточная строка \*)

L11:STRING(16); (\* Промежуточная строка \*)

L12:STRING(16); (\* Промежуточная строка \*)

L13:STRING(16); (\* Промежуточная строка \*)

L14:STRING(16); (\* Промежуточная строка \*)

L15:STRING(16); (\* Промежуточная строка \*)

L32:STRING(32);

L48:STRING(48);

L64:STRING(64);

L80:STRING(80);

L96:STRING(96);

L112:STRING(112);

L128:STRING(128);

L144:STRING(144);

L160:STRING(160);

L176:STRING(176);

L192:STRING(192);

L208:STRING(208);

L224:STRING(224);

L240:STRING(240);

S\_B\_A\_1: S\_B\_A;

Embedment: DWORD; (\* Заглубление, число импульсов \*)

Accel\_1: DWORD; (\* Ускорение по Y \*)

Accel\_m: DWORD := 16000000; (\* Максимальное ускорение по Y \*)

Dir\_1: BOOL; (\* Направление движения по оси Y \*)

Quantity\_1: DWORD; (\* Количество импульсов по оси Y для ШД1 \*)

Quantity\_3: DWORD; (\* Количество импульсов по оси X для ШД3 \*)

Step\_TIME: REAL := 0.006; (\* Время отработк шага по оси Х, сек \*)

Step\_TIME1: REAL := 0.006; (\* Увеличенное время отработк шага по оси Х, сек \*)

P\_D1: REAL := 0.01; (\* Цена дискреты, 1 в массиве MAS[] означает перемещение на 0,01 мм по оси Y \*)

(\*P\_D3: REAL := 0.1; (\* Цена дискреты, 0,1 мм по оси X \*)\*)

P\_D3: REAL := 2.0; (\* Цена дискреты, 2.0 мм по оси X \*)

NUMBER\_I1: WORD := 200; (\* Число импульсов на 1 оборот вала ШД1 \*)

NUMBER\_I3: WORD := 200; (\* Число импульсов на 1 оборот вала ШД3 \*)

TON2: TON ;

PP\_X: REAL := 5.0; (\* Шаг винта по оси Х - 5 мм/об \*)

FL\_STEP1: BOOL; (\* Флаг отработки перемещения по оси Y \*)

FL\_STEP3: BOOL; (\* Флаг отработки перемещения по оси X \*)

COUN: DWORD; (\* Число импульсов, отработанное ШД3 \*)

END\_VAR

### Файл init

Y5 := FALSE;

Y6 := FALSE;

Y7 := FALSE;

Y8 := FALSE;

Y9 := FALSE;

Y10 := FALSE;

Y11 := FALSE;

Y12 := FALSE;

FL1 := FALSE;

E\_I1 := FALSE;

E\_I2 := FALSE;

P\_Y := FALSE;

state := 65;

R10 := 120;

R11 := 120;

R12 := 0;

Reset\_R;

BIT\_28 := BIT\_29 := BIT\_30 := BIT\_31 := BIT\_32 := BIT\_34 := FALSE;

RG3 := NFr;

RG4 := KFr;

RG5 := VFr;

HFr := RG6;

RG7 := IF\_S;

RG8 := Xn1Fr;

RG9 := Xn2Fr;

RG10 := Xn3Fr;

RG11 := Xn4Fr;

RG12 := Xn5Fr;

RG13 := Xk1Fr;

RG14 := Xk2Fr;

RG15 := Xk3Fr;

RG16 := Xk4Fr;

RG17 := Xk5Fr

SOM := FALSE;

SteppersConfig\_Pru0Init(); (\* загрузка кода в PRU0, PRU запущены, можно обмениваться данными \*)

SteppersConfig\_Pru1Init(); (\* загрузка кода в PRU1, PRU запущены, можно обмениваться данными \*)

SetIRQ(200); (\* Разрешаем работу таймера прерывания с периодом 200 мкс \*)

HD1\_STOP;

HD3\_STOP;

### Файл Read\_Prog

(\* Загрузка программы \*)

IF ((BIT\_21) AND (state = 65)) THEN NAME := 'usb:p1.txt'; RG20 := 1; state := 3; END\_IF

IF ((BIT\_22) AND (state = 65)) THEN NAME := 'usb:p2.txt'; RG20 := 2; state := 3; END\_IF

IF ((BIT\_23) AND (state = 65)) THEN NAME := 'usb:p3.txt'; RG20 := 3; state := 3; END\_IF

IF ((BIT\_24) AND (state = 65)) THEN NAME := 'usb:p4.txt'; RG20 := 4; state := 3; END\_IF

IF ((BIT\_25) AND (state = 65)) THEN NAME := 'usb:p5.txt'; RG20 := 5; state := 3; END\_IF

IF ((BIT\_26) AND (state = 65)) THEN NAME := 'usb:p6.txt'; RG20 := 6; state := 3; END\_IF

CASE state OF

3: (\* Пауза \*)

TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)

IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); FL1 := FALSE; BIT\_3 := FALSE; state := 5; END\_IF

5: (\* Открытие файл на чтение, этап 1 \*)

res:=OwenFileOpenAsync(NAME, 'r', ADR(handle));

IF res = ASYNC\_WORKING THEN

N\_C := 0;

P\_E := FALSE;

state := 10; (\* Нет ошибок, переход на открытие файла, этап 2 \*)

END\_IF

10:(\* Открытие файла на чтение, этап 2 \*)

res:=OwenFileOpenAsync(NAME, 'r', ADR(handle));

IF res=ASYNC\_DONE THEN

IF handle <> 0 THEN

state := 20; (\* Нет ошибок, переход на чтение файла, этап 1 \*)

ELSE

(\*state := 5; Возврат на открытие файла, этап 1 \*)

state := 70;

END\_IF

ELSIF res < 0 THEN

(\*state := 5;\*)

state := 70;

END\_IF

20: (\* Чтение файла, этап 1 \*)

res:=OwenFileReadAsync(handle,ADR(bufin),4,ADR(result)); (\* Читаем 4 байта \*)

IF res = ASYNC\_WORKING THEN

state := 30; (\* Нет ошибок, переход на чтение файла, этап 2\*)

ELSE

state:=40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)

END\_IF

30:(\* Чтение файла, этап 2\*)

res:=OwenFileReadAsync(handle, ADR(bufin), 4, ADR(result));

IF res=ASYNC\_DONE THEN

IF result >= 0 THEN (\* Если нет ошибок, и количество считанных данных >=0, то перейти на преобразование строки в байт и заполнение массива \*)

state := 35;

ELSE

state := 40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)

END\_IF

ELSIF res < 0 THEN

state := 40;

END\_IF

35: (\* Преобразование строки в байт и заполнение массива \*)

STR1 := RIGHT(bufin, 1); (\* Выделяем последний символ в строке \*)

STR2 := LEFT(bufin, 3); (\* Отсекаем последний символ в строке \*)

MAS[N\_C] := DEC\_TO\_BYTE(STR2); (\* Преобразуем строку в байт и помещаем в массив \*)

IF (STR1 = '\*') THEN N\_E := N\_C; N\_C := 0; state := 40; ELSE N\_C := N\_C + 1; state := 20; END\_IF

40: (\* Закрытие файл, этап 1 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_WORKING THEN

state := 50; (\* Нет ошибок, переход на закрытие файла, этап 2 \*)

ELSE

state := 60; (\* \*)

END\_IF

50:(\* Закрытие файла, этап 2 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_DONE THEN P\_E := TRUE;

IF result = 0 THEN

state := 60; (\* \*)

ELSE

state := 60; (\* \*)

END\_IF

ELSIF res < 0 THEN

state := 60;

END\_IF

60:(\* Файл загружен успешно \*)

FL1 := TRUE; BIT\_3 := TRUE; state := 65;

65:(\* Ожидание \*)

;

70: (\* Ошибка чтения файла программы \*)

BIT\_0 := TRUE; RG20 := 0; state := 65;

END\_CASE

### Файл Passage\_1

CASE R1 OF

0:

IF ((BIT\_16) AND (FL1 = TRUE) AND (NOT S\_B\_A\_1.B0) AND (NOT SOM)) THEN

Stop\_Motor();

P\_N := TRUE;

SOM := TRUE;

R100 := 0;

R200 := 0;

TON2(IN := FALSE, PT:= T#0s);

FACTOR1 := (P\_D1 \* NUMBER\_I1 \* Kdr1)/STEP\_S\_Y;

Quantity\_3 := REAL\_TO\_DWORD((P\_D3 \* NUMBER\_I3 \* Kdr3)/STEP\_S\_X);

R1 := 5;

END\_IF

5:

initial\_position\_Y;

IF (SSP\_Y = TRUE ) THEN R1 := 10; END\_IF

10:

initial\_position\_X;

IF (SSP\_X = TRUE ) THEN R1 := 20; END\_IF

20:

(\*M1 := TRUE; HD2\_1000; R1 := 30;\*)

M1 := TRUE; R1 := 30;

30: TON2(IN := TRUE, PT:= UT\_2S);

IF (TON2.Q=TRUE) THEN TON2(IN := FALSE, PT:= T#0s); Embedment := REAL\_TO\_DWORD(FACTOR1 \* 100 \* MAS[0]); R1 := 35; END\_IF

35: Stepper\_1(Task\_Speed := 55000, Accel := 50000, Quantity := Embedment, Dir := FALSE);

IF SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_state = STOP\_STEPPER\_RUN\_STATE THEN

SteppersConfig\_Pru1MemoryTransfer.STEPPER1\_PRU1\_stepper\_enable := FALSE; N\_C := 1; R1 := 40; END\_IF

40:Stepper\_3(EN := TRUE, RES := FALSE, PERIOD := 100, DURABILITY := 50); Dir\_3 := TRUE;

FL\_STEP1 := FALSE; FL\_STEP3 := FALSE;

R1 := 45;

45:IF (FL\_STEP3 = TRUE) THEN N\_C := N\_C + 1; R1 := 50; END\_IF

50:

IF (N\_C > N\_E) THEN R1 := 60; ELSE R1 := 40; END\_IF

60:

M1 := FALSE; HD3\_STOP; SOM := FALSE; R200 := 0; R1 := 70;

70: initial\_position\_Y;

IF (SSP\_Y = TRUE) THEN SOM := FALSE; R1 := 0; END\_IF

END\_CASE

### Файл Passage\_2

(\* Обработка за 2 прохода \*)

CASE R2 OF

0: (\* Ожидание \*)

IF ((BIT\_17) AND (FL1 = TRUE) AND (NOT S\_B\_A\_1.B0) AND (NOT SOM)) THEN

Stop\_Motor();

P\_N := TRUE; (\* Номер прохода 1 \*)

C := 0; (\* Обнулили показания энкодера \*)

SOM := TRUE;

R2 := 5;

END\_IF

5: (\* Исходное положение \*)

initial\_position\_Y;

IF (SSP\_Y) THEN R2 := 20; END\_IF

20: (\* Включаем мотор пилы \*)

M1 := TRUE; R2 := 40;

40: (\* Включаем мотор-редуктор привода ХВ - вращение вперёд \*)

M3\_1(F := 50, D := FALSE); E\_I1 := TRUE; P\_Y := FALSE; N\_C := 0; R2 := 50;

50: (\* Ждем окончания прохода \*)

IF (P\_Y = TRUE) THEN R2 := 55; END\_IF

55: (\* Выключаем мотор-редуктор привода ХВ \*)

M3\_1(F := 0); E\_I1 := TRUE; R2 := 57;

60: (\* Включаем мотор-редуктор привода ХВ - вращение назад \*)

M3\_1(F := 25, D := TRUE); E\_I2 := TRUE; P\_Y := FALSE; P\_N := FALSE; N\_C := N\_E; R2 := 70;

70: (\* Ждем окончания прохода 2 \*)

IF (P\_Y = TRUE) THEN R2 := 80; END\_IF

80: (\* Выключаем М1, М2, М3 \*)

(\*M1:=M2\_1:=FALSE; M3\_1(F := 0); R2 := 90;\*)

;

90: (\* Переводим инструмент в нулевое положение \*)

IF (NOT D1) THEN BLINKER; OUT.2 := FALSE; OUT.0 := OUT\_B; SysPortOut(0, OUT); ELSE SOM := FALSE; R2 := 0; END\_IF

END\_CASE;

### Файл R\_M\_P\_S

(\* Чтение параметров с внешней FLASH при подаче питания на ПЛК \*)

CASE R12 OF

0: (\* Сброс \*)

FL2 := FALSE; BIT\_4 := FALSE; BIT\_1 := FALSE; R12 := 3;

3: (\* Пауза \*)

TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)

IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R12 := 5; END\_IF

5: (\* Открытие файл на чтение, этап 1 \*)

(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)

res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));

IF res = ASYNC\_WORKING THEN

R12 := 10; (\* Нет ошибок, переход на открытие файла, этап 2 \*)

END\_IF

10:(\* Открытие файла на чтение, этап 2 \*)

(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)

res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));

IF res=ASYNC\_DONE THEN

IF handle <> 0 THEN

R12 := 20; (\* Нет ошибок, переход на чтение файла, этап 1 \*)

ELSE

R12 := 130;

END\_IF

ELSIF res < 0 THEN

R12 := 130;

END\_IF

20: (\* Чтение файла, этап 1 \*)

res:=OwenFileReadAsync(handle,ADR(L100), 240, ADR(result)); (\* Читаем 4 байта \*)

IF res = ASYNC\_WORKING THEN

R12 := 30; (\* Нет ошибок, переход на чтение файла, этап 2\*)

ELSE

R12:=40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)

END\_IF

30:(\* Чтение файла, этап 2\*)

res:=OwenFileReadAsync(handle, ADR(L100), 240, ADR(result));

IF res=ASYNC\_DONE THEN

IF result >= 0 THEN (\* Если нет ошибок, и количество считанных данных >=0, то перейти на закрытие файла \*)

R12 := 40;

ELSE

R12 := 40; (\* Есть ошибки, переход на закрытия файла, 1 этап \*)

END\_IF

ELSIF res < 0 THEN

R12 := 40;

END\_IF

40: (\* Закрытие файл, этап 1 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_WORKING THEN

R12 := 50; (\* Нет ошибок, переход на закрытие файла, этап 2 \*)

ELSE

R12 := 130; (\* \*)

END\_IF

50:(\* Закрытие файла, этап 2 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_DONE THEN

IF result = 0 THEN

R12 := 100; (\* \*)

ELSE

R12 := 130; (\* \*)

END\_IF

ELSIF res < 0 THEN

R12 := 130;

END\_IF

100:

L1 := MID (L100, 16, 1);

L2 := MID (L100, 16, 17);

L3 := MID (L100, 16, 33);

L4 := MID (L100, 16, 49);

L5 := MID (L100, 16, 65);

L6 := MID (L100, 16, 81);

L7 := MID (L100, 16, 97);

L8 := MID (L100, 16, 113);

L9 := MID (L100, 16, 129);

L10 := MID (L100, 16, 145);

L11 := MID (L100, 16, 161);

L12 := MID (L100, 16, 177);

L13 := MID (L100, 16, 193);

L14 := MID (L100, 16, 209);

L15 := MID (L100, 16, 225);

NFr := RG3 := BIN\_TO\_DWORD(L15);

KFr := RG4 := BIN\_TO\_DWORD(L14);

VFr := RG5 := BIN\_TO\_DWORD(L13);

HFr := RG6 := BIN\_TO\_DWORD(L12);

IF\_S := RG7 := BIN\_TO\_DWORD(L11);

Xn1Fr := RG8 := BIN\_TO\_DWORD(L10);

Xn2Fr := RG9 := BIN\_TO\_DWORD(L9);

Xn3Fr := RG10 := BIN\_TO\_DWORD(L8);

Xn4Fr := RG11 := BIN\_TO\_DWORD(L7);

Xn5Fr := RG12 := BIN\_TO\_DWORD(L6);

Xk1Fr := RG13 := BIN\_TO\_DWORD(L5);

Xk2Fr := RG14 := BIN\_TO\_DWORD(L4);

Xk3Fr := RG15 := BIN\_TO\_DWORD(L3);

Xk4Fr := RG16 := BIN\_TO\_DWORD(L2);

Xk5Fr := RG17 := BIN\_TO\_DWORD(L1);

R12 :=110;

110:(\* Файл загружен успешно \*)

FL2 := TRUE; BIT\_4 := TRUE; R12 :=120;

120:(\* Ожидание \*)

;

130: (\* Ошибка чтения параметров фреэерования \*)

BIT\_1 := TRUE; R12 :=120;

END\_CASE

### Файл R\_M\_P

IF ((BIT\_34) AND (R10 = 120)) THEN R10 := 0; END\_IF

CASE R10 OF

0:

FL2 := FALSE; BIT\_4 := FALSE; BIT\_1 := FALSE; R10 := 3;

3:

TON1(IN := TRUE, PT:= UT\_1S);

IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R10 := 5; END\_IF

5:

(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)

res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));

IF res = ASYNC\_WORKING THEN

R10 := 10;

END\_IF

10:

(\*res:=OwenFileOpenAsync('usb:param.txt', 'r', ADR(handle));\*)

res:=OwenFileOpenAsync(NAME1, 'r', ADR(handle));

IF res=ASYNC\_DONE THEN

IF handle <> 0 THEN

R10 := 20;

ELSE

R10 := 130;

END\_IF

ELSIF res < 0 THEN

R10 := 130;

END\_IF

20:

res:=OwenFileReadAsync(handle,ADR(L100), 240, ADR(result));

IF res = ASYNC\_WORKING THEN

R10 := 30;

ELSE

R10:=40;

END\_IF

30:

res:=OwenFileReadAsync(handle, ADR(L100), 240, ADR(result));

IF res=ASYNC\_DONE THEN

IF result >= 0 THEN

R10 := 40;

ELSE

R10 := 40;

END\_IF

ELSIF res < 0 THEN

R10 := 40;

END\_IF

40:

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_WORKING THEN

R10 := 50;

ELSE

R10 := 130; (\* \*)

END\_IF

50:

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_DONE THEN

IF result = 0 THEN

R10 := 100; (\* \*)

ELSE

R10 := 130; (\* \*)

END\_IF

ELSIF res < 0 THEN

R10 := 130;

END\_IF

100:

L1 := MID (L100, 16, 1);

L2 := MID (L100, 16, 17);

L3 := MID (L100, 16, 33);

L4 := MID (L100, 16, 49);

L5 := MID (L100, 16, 65);

L6 := MID (L100, 16, 81);

L7 := MID (L100, 16, 97);

L8 := MID (L100, 16, 113);

L9 := MID (L100, 16, 129);

L10 := MID (L100, 16, 145);

L11 := MID (L100, 16, 161);

L12 := MID (L100, 16, 177);

L13 := MID (L100, 16, 193);

L14 := MID (L100, 16, 209);

L15 := MID (L100, 16, 225);

NFr := RG3 := BIN\_TO\_DWORD(L15);

KFr := RG4 := BIN\_TO\_DWORD(L14);

VFr := RG5 := BIN\_TO\_DWORD(L13);

HFr := RG6 := BIN\_TO\_DWORD(L12);

IF\_S := RG7 := BIN\_TO\_DWORD(L11);

Xn1Fr := RG8 := BIN\_TO\_DWORD(L10);

Xn2Fr := RG9 := BIN\_TO\_DWORD(L9);

Xn3Fr := RG10 := BIN\_TO\_DWORD(L8);

Xn4Fr := RG11 := BIN\_TO\_DWORD(L7);

Xn5Fr := RG12 := BIN\_TO\_DWORD(L6);

Xk1Fr := RG13 := BIN\_TO\_DWORD(L5);

Xk2Fr := RG14 := BIN\_TO\_DWORD(L4);

Xk3Fr := RG15 := BIN\_TO\_DWORD(L3);

Xk4Fr := RG16 := BIN\_TO\_DWORD(L2);

Xk5Fr := RG17 := BIN\_TO\_DWORD(L1);

R10 :=110;

110:

FL2 := TRUE; BIT\_4 := TRUE; R10 :=120;

120:

;

130:

BIT\_1 := TRUE; R10 :=120;

END\_CASE

### Файл W\_M\_P

(\* Запись параметров фрезерования \*)

IF ((BIT\_35) AND (R11 = 120)) THEN R11 := 0; END\_IF

CASE R11 OF

0: (\* Сброс \*)

FL2 := FALSE; BIT\_5 := FALSE; BIT\_2 := FALSE; R11 := 3;

3: (\* Пауза \*)

TON1(IN := TRUE, PT:= UT\_1S); (\* Запустили таймер \*)

IF TON1.Q=TRUE THEN TON1(IN := FALSE, PT:= T#0s); R11 := 5; END\_IF

5: (\* \*)

NFr := RG3;

KFr := RG4;

VFr := RG5;

HFr := RG6;

IF\_S := RG7;

Xn1Fr := RG8;

Xn2Fr := RG9;

Xn3Fr := RG10;

Xn4Fr := RG11;

Xn5Fr := RG12;

Xk1Fr := RG13;

Xk2Fr := RG14;

Xk3Fr := RG15;

Xk4Fr := RG16;

Xk5Fr := RG17;

L1 := DWORD\_TO\_STRB(NFr);

L2 := DWORD\_TO\_STRB(KFr);

L3 := DWORD\_TO\_STRB(VFr);

L4 := DWORD\_TO\_STRB(HFr);

L5 := DWORD\_TO\_STRB(IF\_S);

L6 := DWORD\_TO\_STRB(Xn1Fr);

L7 := DWORD\_TO\_STRB(Xn2Fr);

L8 := DWORD\_TO\_STRB(Xn3Fr);

L9 := DWORD\_TO\_STRB(Xn4Fr);

L10 := DWORD\_TO\_STRB(Xn5Fr);

L11 := DWORD\_TO\_STRB(Xk1Fr);

L12 := DWORD\_TO\_STRB(Xk2Fr);

L13 := DWORD\_TO\_STRB(Xk3Fr);

L14 := DWORD\_TO\_STRB(Xk4Fr);

L15 := DWORD\_TO\_STRB(Xk5Fr);

L32 := CONCAT(L2, L1);

L48 := CONCAT(L3, L32);

L64 := CONCAT(L4, L48);

L80 := CONCAT(L5, L64);

L96 := CONCAT(L6, L80);

L112 := CONCAT(L7, L96);

L128 := CONCAT(L8, L112);

L144 := CONCAT(L9, L128);

L160 := CONCAT(L10, L144);

L176 := CONCAT(L11, L160);

L192 := CONCAT(L12, L176);

L208 := CONCAT(L13, L192);

L224 := CONCAT(L14, L208);

L240 := CONCAT(L15, L224);

R11 := 50;

50: (\* Открываем файл на запись, этап 1 \*)

res:=OwenFileOpenAsync(NAME1,'w',ADR(handle)); (\* Возвращает дескриптор файла, либо 0(в некоторых случаях 4 294 967 295 или -1) в случае ошибки)\*)

IF res=ASYNC\_WORKING THEN

R11 := 55;(\* Нет ошибок, переход на открытие файла, этап 1 \*)

END\_IF

55: (\* Открываем файл, этап 2 \*)

res:=OwenFileOpenAsync(NAME1,'w',ADR(handle));

IF res=ASYNC\_DONE THEN (\* Если "готов", и нет ошибок \*)

IF handle<>0 THEN

R11 := 60; (\* Нет ошибок, переход на запись этап 1\*)

ELSE

R11 := 130; (\* \*)

END\_IF

ELSIF res < 0 THEN

R11 := 130;

END\_IF

60: (\* Пишем файл, этап 1 \*)

res:=OwenFileWriteAsync(handle,ADR(L240), LEN(L240), ADR(result));

IF res=ASYNC\_WORKING THEN

R11 := 65; (\* Нет ошибок, переход на запись файла, этап 2 \*)

ELSE

R11 := 70; (\* Иначе переход на этап закрытия файла \*)

END\_IF

65: (\* Пишем в файл, этап 2\*)

res:=OwenFileWriteAsync(handle, ADR(L240), LEN(L240), ADR(result));

IF res=ASYNC\_DONE THEN

IF (result = LEN(L240)) THEN (\* Если результат равен длине строки, то переход на закрытие файла \*)

R11 := 70;

ELSE

BIT\_2 := TRUE; R11 := 70; (\* Иначе переход на этап закрытия файла \*)

END\_IF

ELSIF res<0 THEN

BIT\_2 := TRUE; R11 := 70;

END\_IF

70: (\* Закрываем файл, этап 1 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_WORKING THEN

R11 := 80; (\*Нет ошибок, переход на закрытие файла этап 2\*)

ELSE

BIT\_2 := TRUE; R11 := 80;

END\_IF

80: (\* Закрываем файл, этап 2 \*)

res:=OwenFileCloseAsync(handle, ADR(result));

IF res=ASYNC\_DONE THEN

IF result=0 THEN

R11 := 110; (\* Файл закрыт успешно \*)

ELSE

R11 := 110;

END\_IF

ELSIF res < 0 THEN

R11 := 110;

END\_IF

110: (\* Файл загружен успешно \*)

FL2 := TRUE; BIT\_5 := TRUE; R11 :=120;

120:(\* Ожидание \*)

;

130: (\* Ошибка чтения параметров фреэерования \*)

BIT\_2 := TRUE; R11 :=120;

END\_CASE

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в докум. | № докум. | Входящий  № сопроводи- тельного докум. и  дата | Подп. | Дата |
| изменен- ных | заменен- ных | новых | аннулиро- ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

УТВЕРЖДАЮ

Зав. кафедрой К-3 МФ

А.В.Корольков.

« » 2025 г.

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Руководство оператора

ЛИСТ УТВЕРЖДЕНИЯ

643.МГУЛ.090301-45 34 01-ЛУ

Студент

М.Ю.Аленина

« » 2025 г.

Руководитель

А.В. Афанасьев

« » 2025 г.

Нормоконтролер

А.В. Чернышов

« » 2025 г.

УТВЕРЖДЕН

643.МГУЛ.090301-45 34 01-ЛУ

Программное обеспечение для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением

Руководство оператора

643.МГУЛ.090301-45 34 01

Листов 6

## АННОТАЦИЯ

В данном программном документе приведено руководство оператора к программному обеспечению для деревообрабатывающего токарно-фрезерного станка с числовым программным управлением.

## СОДЕРЖАНИЕ

1. Назначение программы 4
2. Условия выполнения программы 4
3. Выполнение программы 5
4. Сообщения оператору 6

### 1 Назначение программы

Программное обеспечение разработано для управления деревообрабатывающим токарно-фрезерным станком с ЧПУ. Основные функции ПО включают:

* **Автоматизация процесса обработки:** Выполнение токарных и фрезерных операций по заранее заданной программе.
* **Управление движением инструмента:** Координация работы шаговых двигателей для перемещения инструмента по заданным траекториям.
* **Управление вращением заготовки:** Контроль скорости и направления вращения заготовки.
* **Управление двигателями:** Включение и выключение шпинделя, фрезы и других двигателей станка.
* **Мониторинг состояния:** Отображение информации о текущих координатах инструмента, скорости вращения шпинделя, состоянии датчиков и других параметрах.
* **Обеспечение безопасности:** Автоматическая остановка станка в случае возникновения аварийных ситуаций.
* **Загрузка данных:** Чтение параметров обработки и управляющей программы из внешних источников (файлов на USB-накопителе).
* **Ручное управление:** Предоставление возможности оператору вручную управлять перемещением инструмента и вращением заготовки для наладки и настройки станка.

### 2 Условия выполнения программы

Для корректной работы программного обеспечения необходимо соблюдение следующих условий:

* **Подключенное оборудование:** Токарно-фрезерный станок с ЧПУ должен быть подключен к управляющему компьютеру (ПЛК) и настроен в соответствии с инструкцией по эксплуатации станка.
* **Питание:** Все компоненты системы управления (станок, ПЛК, панель оператора) должны быть подключены к источнику питания и находиться во включенном состоянии.
* **Установленное ПО:** На управляющем компьютере (ПЛК) должно быть установлено и корректно сконфигурировано программное обеспечение управления станком.
* **Аварийные выключатели:** Убедитесь, что все аварийные выключатели (кнопка “СТОП”) находятся в исходном положении (не нажаты).
* **Безопасность:** Оператор должен ознакомиться с инструкцией по технике безопасности при работе с токарно-фрезерным станком с ЧПУ и соблюдать все необходимые меры предосторожности.
* **Исходное положение:** Убедитесь, что станок находится в исходном положении (инструмент отведен, заготовка зафиксирована) перед запуском программы.

### 3 Выполнение программы

Порядок выполнения программы управления станком:

1. Включение системы: Включите станок и управляющий компьютер (ПЛК).
2. Запуск программного обеспечения: Запустите программу управления станком на панели оператора.
3. Выбор режима работы: На панели оператора выберите необходимый режим работы (например, “Токарная обработка”, “Фрезерная обработка”, “Ручное управление”).
4. Загрузка данных (если необходимо): Если требуется, загрузите параметры обработки из файла на USB-накопителе. Используйте для этого функцию “Загрузка” на панели оператора.
5. Настройка параметров: При необходимости, настройте параметры обработки (скорость вращения шпинделя, скорость подачи, глубину резания и т.д.) на панели оператора.
6. Проверка программы: Перед запуском программы убедитесь в правильности выбранных параметров и траектории движения инструмента.
7. Запуск обработки: Нажмите кнопку “Старт” на панели оператора.
8. Контроль процесса: Во время выполнения программы следите за показаниями на панели оператора и контролируйте процесс обработки. В случае возникновения нештатной ситуации немедленно нажмите кнопку “СТОП”.
9. Завершение программы: После завершения программы убедитесь, что станок остановился, а инструмент находится в безопасном положении.
10. Выключение системы: Выключите станок и управляющий компьютер (ПЛК).

### 4 Сообщения оператору

При запуске любой из программ, на экране отображается заставка соответствующей программы.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в докум. | № докум. | Входящий  № сопроводительного докум. и  дата | Подп. | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |