А. Л. ЗОЛКИН

ПРОГРАММИРОВАНИЕ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ

УЧЕБНИК

ЛАНЬ
САНКТ-ПЕТЕРБУРГ•МОСКВА•КРАСНОДАР
2025

УДК 004.77
ББК 32.973.202я723

З 79 **Золкин А. Л.** Программирование логических контроллеров : учебник для СПО / А. Л. Золкин. — Санкт-Петербург : Лань, 2025. — 148 с. : ил. — Текст : непосредственный.

ISBN 978-5-507-51614-8

Целью учебника является предоставить комплексное руководство по программированию логических контроллеров (PLC) для студентов технических специальностей, включая их роль, принципы работы и ключевые компоненты. В первой и второй главах подробно описаны архитектура систем сбора данных, функции и задачи диспетчеризации, а также ключевые компоненты, такие как сенсоры, преобразователи, контроллеры и интерфейсы пользователя. Рассматриваются типовые схемы взаимодействия компонентов и примеры использования систем в различных отраслях промышленности. В третьей и четвертой главах учебника основное внимание уделяется разработке программ для PLC и использованию программных сред и инструментов.

Учебник предназначен для изучения дисциплины «Информатика» студентами колледжей специальностей направлений подготовки «Информатика и вычислительная техника», «Электроника, радиотехника и системы связи», «Информационная безопасность». Соответствует современным требованиям Федерального государственного образовательного стандарта среднего профессионального образования и профессиональным квалификационным требованиям.

УДК 004.77
ББК 32.973.202я723

Рецензенты:

А. П. ДОЛГИНЦЕВ — кандидат технических наук, доцент, доцент кафедры цифровых технологий Приволжского государственного университета путей сообщения;

Р. С. ЗАРИПОВА — кандидат технических наук, доцент кафедры цифровых систем и моделей Казанского государственного энергетического университета.

Сведения об авторе

Золкин Александр Леонидович — кандидат технических наук, доцент, доцент кафедры «Информатика и вычислительная техника», ФГБОУ ВО «Поволжский государственный университет телекоммуникаций и информатики» (ПГУТИ), г. Самара, Россия.

Обложка
П. И. ПОЛЯКОВА

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Логические контроллеры (PLC) занимают ключевое место в современных системах автоматизации. Они используются для управления технологическими процессами, обеспечения надежности и безопасности производственных систем, а также для повышения эффективности работы оборудования. В последние десятилетия промышленная автоматизация стала неотъемлемой частью многих отраслей — от производства и энергетики до транспорта и строительства. Именно поэтому владение навыками программирования логических контроллеров является важным требованием для специалистов, работающих в этих сферах [1, 14].

Целью данного учебника является предоставление комплексного и систематического руководства по программированию логических контроллеров. В нем рассматриваются как базовые концепции и теоретические основы, так и продвинутые методики разработки программ, отладки и тестирования систем автоматизации. Особое внимание уделяется языкам программирования PLC, таким как Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) и Sequential Function Chart (SFC).

Учебник предназначен для студентов технических специальностей, инженеров по автоматизации и всех, кто стремится углубить свои знания в области промышленной автоматизации. Структура книги построена таким образом, чтобы обучающиеся могли последовательно освоить материал, начиная с основ и постепенно переходя к более сложным темам. Каждый раздел снабжен примерами и практическими заданиями, что способствует лучшему усвоению информации и развитию практических навыков.

Современные технологии и программные средства для разработки и симуляции программ для логических контроллеров также рассматриваются в данном учебнике. Это включает в себя такие инструменты, как Siemens TIA Portal, Rockwell Automation Studio 5000 и Schneider Electric EcoStruxure.

Обучающиеся узнают, как использовать эти среды для создания, отладки и тестирования программ, что является важным навыком для работы в реальных условиях.

Введение также акцентирует внимание на важности постоянного обучения и профессионального развития в быстро меняющемся мире технологий. Программирование логических контроллеров — это область, которая требует не только теоретических знаний, но и постоянной практики и адаптации к новым инструментам и методологиям. Надеемся, что данный учебник станет надежным спутником и по-

мощником в этом процессе, помогая обучающимся достигать новых высот в их профессиональной деятельности.

Автоматизация производственных процессов с использованием логических контроллеров позволяет существенно повысить производительность и качество продукции, снизить затраты на обслуживание оборудования и минимизировать человеческий фактор. Введение PLC-систем в производство позволяет автоматизировать сложные и многоступенчатые процессы, обеспечивая их точное и надежное выполнение. Благодаря своим преимуществам логические контроллеры находят широкое применение в различных отраслях промышленности, таких как машиностроение, химическая промышленность, пищевая промышленность и энергетика.

Важным аспектом использования логических контроллеров является их способность интегрироваться с другими системами автоматизации, такими как SCADA и MES. Это позволяет создавать комплексные системы управления, охватывающие все уровни производственного процесса — от управления отдельными машинами до координации работы целых производственных линий и заводов. В учебнике рассматриваются принципы интеграции и взаимодействия PLC с другими системами, что является ключевым фактором для успешного внедрения автоматизированных решений.

В ходе написания учебника автор стремился учитывать актуальные требования и тенденции в области промышленной автоматизации.

Это включает в себя не только традиционные подходы и методы, но и новые технологии, такие как Интернет вещей (IoT), промышленный Интернет вещей (IIoT), искусственный интеллект (AI) и машинное обучение (ML). Особое внимание уделяется вопросам кибербезопасности в автоматизированных системах, поскольку защита данных и предотвращение несанкционированного доступа становятся все более важными в современном цифровом мире.

Учебник «Программирование логических контроллеров» также направлен на развитие критического мышления и творческого подхода к решению задач автоматизации. Обучающиеся научатся анализировать требования и ограничения конкретных производственных процессов, разрабатывать оптимальные решения с использованием логических контроллеров и оценивать их эффективность. Практические задания и кейсы, представленные в учебнике, способствуют развитию навыков проектирования и реализации комплексных систем автоматизации.

В заключение хотелось бы отметить, что успешное освоение программирования логических контроллеров открывает широкие

возможности для профессионального роста и карьерного развития. В мире, где автоматизация играет все большую роль, специалисты, обладающие глубокими знаниями и практическими навыками в этой области, всегда будут востребованы. Мы надеемся, что данный учебник станет надежным и полезным инструментом для всех, кто стремится к профессиональному совершенствованию и желает внести свой вклад в развитие современных технологий автоматизации.

ГЛАВА 1. ТИПЫ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ И ИХ ПРИМЕНЕНИЕ

1.1. Архитектура PLC: аппаратное и программное обеспечение

Архитектура программируемых логических контроллеров (PLC) включает два основных компонента: аппаратное и программное обеспечение. Аппаратное обеспечение состоит из центрального процессора (CPU), модулей ввода-вывода (I/O), памяти, блоков питания и интерфейсов связи. Центральный процессор выполняет функции обработки данных и управления, осуществляя выполнение программ и управление входными и выходными сигналами. Модули ввода-вывода обеспечивают взаимодействие контроллера с внешними устройствами, такими как датчики и исполнительные механизмы, преобразуя сигналы в форму, пригодную для обработки CPU.

Память PLC разделяется на оперативную память (RAM) и постоянную память (ROM). Оперативная память используется для хранения временных данных и выполнения программ, в то время как постоянная память хранит прошивки и конфигурации, которые остаются неизменными при отключении питания. Блоки питания обеспечивают стабильное напряжение и ток для всех компонентов PLC, что критически важно для надежной работы системы.

Блоки питания в программируемых логических контроллерах (PLC) играют важную роль, обеспечивая стабильное и надежное электропитание всех компонентов системы. Они преобразуют переменное напряжение сети (обычно 220 или 110 В переменного тока) в постоянное напряжение, необходимое для работы контроллера и его модулей. Стабильное напряжение и ток являются критически важными для корректного функционирования PLC, так как даже незначительные колебания могут привести к сбоям или некорректной работе системы [2, 9].

Современные блоки питания для PLC оснащены функциями защиты от перенапряжения, перегрузки по току и короткого замыкания. Это означает, что в случае возникновения электрических аномалий блок питания может отключить подачу питания на контроллер, предотвращая повреждение оборудования и обеспечивая безопасность системы.

Некоторые блоки питания также имеют функции резервирования и горячей замены, что позволяет заменять их без остановки рабо-

ты всей системы, повышая надежность и доступность автоматизированного процесса.

Некоторые блоки питания для программируемых логических контроллеров (PLC) оснащены функциями резервирования и горячей замены, что значительно повышает надежность и доступность автоматизированных систем. Эти функции обеспечивают непрерывность работы и минимизацию простоев, что критически важно для промышленных процессов, где каждая минута простоя может приводить к значительным финансовым потерям.

Функция резервирования заключается в использовании двух или более блоков питания, работающих параллельно. В такой конфигурации один блок питания может полностью обеспечивать систему энергией, в то время как второй служит резервным. В случае выхода из строя основного блока питания резервный блок автоматически берет на себя его функции без прерывания работы системы. Такая схема значительно повышает отказоустойчивость, поскольку вероятность одновременного отказа нескольких блоков питания крайне мала.

Функция горячей замены позволяет заменять блок питания без необходимости отключения всей системы. Это достигается за счет использования специальных разъемов и схемы подключения, которые позволяют безопасно извлечь неисправный блок питания и установить новый, не прерывая при этом подачу энергии к остальным компонентам системы. Важно, чтобы блоки питания и сам контроллер поддерживали эту функцию, обеспечивая безопасную и эффективную замену.

Применение этих технологий значительно улучшает эксплуатационные характеристики системы. В первую очередь это касается надежности — система остается работоспособной даже в случае отказа одного из блоков питания. Во-вторых, это повышает доступность, так как техническое обслуживание и замена компонентов могут проводиться без остановки производственного процесса. В-третьих, такая конфигурация упрощает планирование технического обслуживания, так как замена блоков питания может проводиться в удобное время, без необходимости планирования простоев.

В дополнение к этому блоки питания с функциями резервирования и горячей замены часто оснащены системами мониторинга и диагностики, которые позволяют в режиме реального времени отслеживать их состояние и предсказать возможные отказы.

Эти системы могут интегрироваться с программным обеспечением управления и диспетчеризации, предоставляя операторам пол-

ную информацию о состоянии блоков питания и позволяя оперативно реагировать на любые изменения.

Таким образом, блоки питания с функциями резервирования и горячей замены существенно повышают надежность и эффективность работы PLC-систем, обеспечивая непрерывность процессов и минимизируя риски, связанные с потерей питания. Эти технологии особенно важны для критически важных приложений, где простои недопустимы и могут привести к серьезным последствиям.

Эти технологии особенно важны для критически важных приложений, где простои недопустимы и могут привести к серьезным последствиям. Критически важные приложения встречаются в различных отраслях, таких как энергетика, транспорт, здравоохранение, нефтегазовая промышленность и производство [45–49].

В энергетической отрасли, где логические контроллеры управляют генерацией, передачей и распределением электроэнергии, простои могут привести к перебоям в электроснабжении. Это может затронуть тысячи потребителей, включая предприятия и бытовых клиентов. Система резервирования и горячей замены блоков питания в PLC обеспечивает непрерывное управление энергосистемами, предотвращая аварии и обеспечивая стабильную подачу электроэнергии.

В системах управления железнодорожным транспортом, авиацией и судоходством простои могут привести к серьезным авариям, задержкам и нарушению графика движения. Например, в системах управления железнодорожными светофорами и стрелочными переводами отказ PLC может создать опасные ситуации на железнодорожных путях. Наличие резервных блоков питания и возможность их горячей замены гарантирует, что системы управления транспортом будут работать бесперебойно, обеспечивая безопасность пассажиров и грузов.

В медицинских учреждениях, таких как больницы, системы автоматизации используются для управления жизнеобеспечивающим оборудованием, включая аппараты искусственного дыхания, системы мониторинга пациентов и оборудование для диагностики. Перебои в работе таких систем могут поставить под угрозу жизнь пациентов.

Резервирование и горячая замена блоков питания в системах автоматизации медицинского оборудования обеспечивают надежность и непрерывность работы, что критически важно для здоровья и безопасности пациентов.

В нефтегазовой отрасли логические контроллеры управляют добычей, переработкой и транспортировкой углеводородов. Простои в этих процессах могут привести к значительным финансовым потерям

и экологическим катастрофам, например разливам нефти или взрывам газовых установок. Системы с резервированием и горячей заменой блоков питания помогают минимизировать риски отказа оборудования, обеспечивая непрерывность и безопасность производственных процессов [26, 31, 34].

В производственных цехах, особенно в высокоавтоматизированных и непрерывных производствах, простои могут привести к значительным финансовым потерям из-за остановки производственных линий и порчи продукции. Системы с функциями резервирования и горячей замены блоков питания обеспечивают непрерывную работу производственных процессов, уменьшая время простоя и повышая общую эффективность и производительность предприятия.

Таким образом, технологии резервирования и горячей замены блоков питания в программируемых логических контроллерах играют ключевую роль в обеспечении надежности и непрерывности работы критически важных приложений. Эти технологии помогают предотвратить простои, минимизировать риски и обеспечить безопасность и эффективность различных промышленных и инфраструктурных процессов [50–54].

Кроме того, блоки питания часто имеют индикацию состояния и диагностики, позволяя операторам и техническому персоналу быстро выявлять и устранять проблемы, связанные с электропитанием. Важно учитывать, что различные компоненты PLC могут требовать разные уровни напряжения и тока, поэтому блоки питания могут быть сконфигурированы для подачи нескольких выходных напряжений, таких как 24 В для цифровых входов/выходов или 5 В для логических схем процессора.

Таким образом, блоки питания обеспечивают не только преобразование и стабилизацию электрической энергии, но и защищают всю систему от электрических аномалий, способствуют повышению надежности и долговечности PLC и поддерживают бесперебойную работу автоматизированных процессов.

Интерфейсы связи включают различные порты и протоколы, такие как Ethernet, RS-232, RS-485 и др., которые позволяют PLC взаимодействовать с другими устройствами и системами. Это может включать как простые периферийные устройства, так и сложные сети автоматизации, обеспечивая интеграцию и обмен данными на разных уровнях производства.

Логические контроллеры (PLC) играют ведущую роль в автоматизации производственных процессов, обеспечивая интеграцию и обмен данными на различных уровнях производства. Это включает вза-

имодействие как с простыми периферийными устройствами, так и со сложными сетями автоматизации. Рассмотрим подробнее, как это происходит.

Периферийные устройства включают датчики, исполнительные механизмы, кнопочные панели, световые индикаторы и другие устройства, которые непосредственно взаимодействуют с процессом. Датчики собирают информацию о различных параметрах процесса, таких как температура, давление, уровень жидкости, положение механизма и др. Эти данные поступают на входные модули PLC, где они обрабатываются и используются для принятия решений. Исполнительные механизмы, такие как электромоторы, клапаны и реле, получают команды от выходных модулей PLC для выполнения действий, которые влияют на производственный процесс. Это может быть включение/выключение оборудования, регулирование скорости или изменение направления движения.

В современных производственных системах PLC часто интегрируются в более сложные сети автоматизации, такие как SCADA (Supervisory Control and Data Acquisition) и MES (Manufacturing Execution Systems). SCADA-системы обеспечивают централизованное управление и мониторинг всего производственного процесса, собирая данные от множества PLC и отображая их в режиме реального времени для операторов. MES-системы занимаются управлением производственными операциями, отслеживая выполнение производственных заказов, управление ресурсами и сбор данных для анализа производительности.

На уровне производственной линии несколько PLC могут быть объединены в локальную сеть, обеспечивая координацию и синхронизацию различных участков линии. Это позволяет автоматизировать сложные последовательности операций, такие как сборка, упаковка и контроль качества.

Данные, собранные каждым PLC, могут быть переданы в центральную систему управления, что обеспечивает единое представление о состоянии всей производственной линии.

На уровне предприятия PLC взаимодействуют с системами верхнего уровня, такими как ERP (Enterprise Resource Planning), которые управляют ресурсами всего предприятия, включая материальные запасы, финансовые операции и человеческие ресурсы. Интеграция PLC с ERP-системами позволяет автоматизировать управление производственными процессами в соответствии с бизнес-целями и стратегиями предприятия. Например, данные о производительности оборудования и

использовании материалов, собранные PLC, могут быть переданы в ERP-систему для анализа и принятия управленческих решений.

Для обеспечения эффективной интеграции и обмена данными PLC поддерживают различные коммуникационные протоколы, такие как Modbus, Profibus, Ethernet/IP, Profinet и др. Эти протоколы стандартизируют передачу данных между устройствами, обеспечивая совместимость и надежность обмена информацией. Применение сетевых технологий, таких как промышленный Ethernet, позволяет создавать масштабируемые и гибкие сети, которые легко адаптируются к изменяющимся требованиям производства.

Применение сетевых технологий, таких как промышленный Ethernet, играет ключевую роль в создании масштабируемых и гибких сетей в промышленной автоматизации. Вот некоторые аспекты, почему промышленный Ethernet стал предпочтительным выбором для многих предприятий:

Промышленный Ethernet предлагает высокую пропускную способность, что позволяет передавать большие объемы данных между устройствами на производственной линии. Это особенно важно в современных производственных средах, где устройства генерируют огромные объемы данных, такие как изображения с камер видеонаблюдения, сигналы с датчиков и информация о состоянии оборудования. Низкая задержка обеспечивает точность и надежность выполнения операций в режиме реального времени. Промышленный Ethernet поддерживает гибридные сетевые топологии, такие как звезда, кольцо и дерево, а также возможность создания виртуальных сегментов сети (VLAN), что обеспечивает гибкость в проектировании и настройке сетей под конкретные потребности производства. Это позволяет легко масштабировать сеть при изменении объемов производства или добавлении новых устройств [21, 27].

Отказоустойчивость. Промышленный Ethernet часто предлагает механизмы отказоустойчивости, такие как протоколы связующего дерева (Spanning Tree Protocol) или кольцевая топология с механизмами быстрого восстановления. Это позволяет обеспечить высокую доступность сети и предотвратить простои в случае отказа устройств или линий связи (рис. 1).

Промышленный Ethernet основан на широко распространенном стандарте Ethernet, что обеспечивает совместимость с широким спектром сетевых устройств и компонентов. Это упрощает интеграцию существующего оборудования и снижает затраты на развертывание новых сетей.

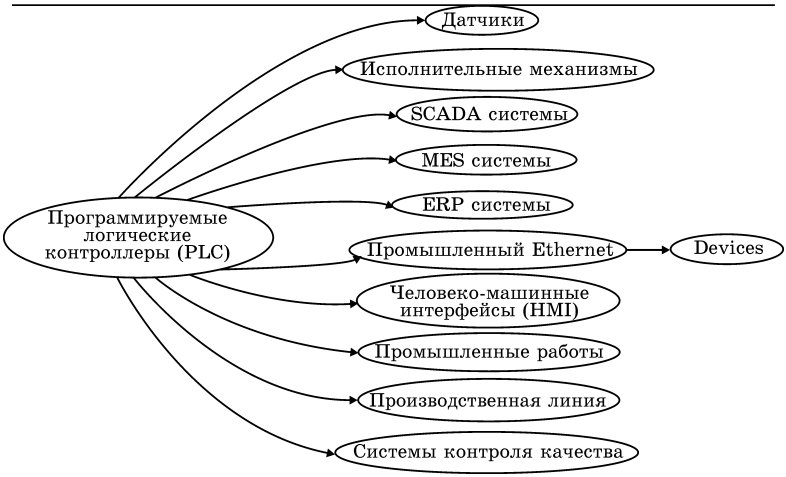

Рис. 1

Схема взаимодействия компонентов в промышленной среде

Промышленный Ethernet поддерживает различные механизмы защиты данных и сетевой инфраструктуры, такие как виртуальные частные сети (VPN), шифрование данных, аутентификация и авторизация пользователей. Это обеспечивает конфиденциальность, целостность и доступность данных в промышленной среде.

Таким образом, промышленный Ethernet представляет собой мощный инструмент для создания высокопроизводительных, отказоустойчивых и безопасных сетей в промышленной автоматизации. Его гибкость, масштабируемость и надежность делают его идеальным выбором для современных производственных сред, где требуется эффективное управление данными и оборудованием.

Таким образом, PLC играют ключевую роль в автоматизации производства, обеспечивая интеграцию и обмен данными на всех уровнях — от простых периферийных устройств до сложных сетей управления предприятием. Это позволяет повысить эффективность, надежность и гибкость производственных процессов, обеспечивая конкурентные преимущества и устойчивое развитие предприятия.

Программное обеспечение PLC состоит из операционной системы, которая управляет базовыми функциями контроллера, и среды разработки приложений, используемой для написания, отладки и загрузки пользовательских программ.

Операционная система контроллера обеспечивает многозадачность, управление памятью и интерфейсами, а также выполнение программ в режиме реального времени (рис. 2).

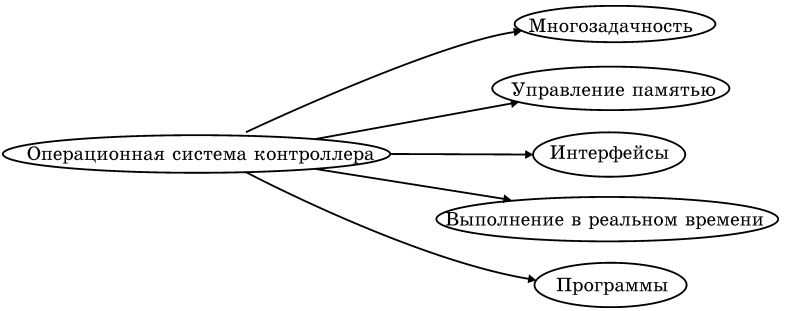

Рис. 2
Функции операционной системы контроллера

Среды разработки, такие как Siemens TIA Portal или Rockwell Automation Studio 5000, предоставляют инструменты для создания и тестирования программ с использованием различных языков программирования, таких как Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) и Sequential Function Chart (SFC) [14, 18].

Архитектура PLC также поддерживает возможность расширения и модернизации. Многие контроллеры имеют модульную структуру, что позволяет добавлять новые модули ввода-вывода, коммуникационные модули или дополнительные процессорные блоки по мере роста и усложнения системы (рис. 3).

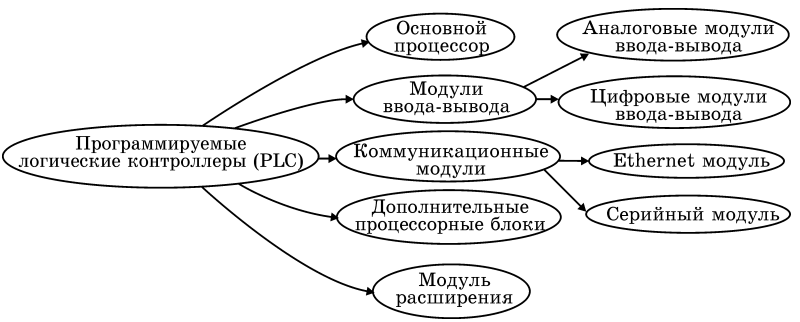

Рис. 3
Схема архитектуры расширяемых и модернизируемых PLC

Это обеспечивает гибкость и адаптивность решений на основе PLC, позволяя им эволюционировать вместе с изменяющимися требованиями производства.

Понимание структуры и функциональности этих компонентов является ключевым для разработки, внедрения и обслуживания систем автоматизации на базе логических контроллеров.

Процессор PLC, модули ввода-вывода, коммуникационные интерфейсы и другие компоненты образуют основу систем автоматизации. Глубокое понимание их структуры и функциональности существенно на всех этапах разработки, внедрения и обслуживания автоматизированных систем [2, 3, 21].

При разработке новых систем важно выбирать подходящие компоненты, учитывая требования конкретного производства и конечные цели автоматизации. Понимание структуры каждого компонента помогает интегрировать их в единую систему, обеспечивая оптимальную производительность и надежность работы.

При внедрении системы необходимо учитывать особенности каждого компонента и его взаимодействия с другими элементами системы. Это позволяет правильно настроить параметры и функции контроллера, обеспечивая эффективное управление производственными процессами.

Во время обслуживания системы важно иметь глубокое понимание структуры и работы компонентов, чтобы оперативно реагировать на любые сбои или неисправности. Это позволяет быстро находить и устранять проблемы, минимизируя простои и обеспечивая непрерывную работу производства.

Таким образом, понимание структуры и функциональности компонентов PLC является ключевым аспектом успешной разработки, внедрения и обслуживания систем автоматизации, обеспечивая их эффективную работу и соответствие потребностям производства [55–59].

1.2. Основные принципы работы и функциональные возможности

Основные принципы работы и функциональные возможности логических контроллеров (PLC) определяют их способность эффективно управлять производственными процессами. В их основе лежит концепция программирования на основе логических схем, где программист создает программы, используя логические операции, такие как И, ИЛИ, НЕ, а также временные и счетные функции. Эти программы определяют поведение контроллера в зависимости от входных сигналов, которые он получает от датчиков и других устройств, и

управляют выходными сигналами, направляемыми к исполнительным механизмам и устройствам в производственной системе [6, 9].

Важным аспектом работы PLC является их способность обрабатывать данные и выполнение программ в режиме реального времени. Это означает, что контроллеры способны реагировать на изменения в окружающей среде и выполнять управляющие действия с минимальной задержкой, что критически важно для обеспечения стабильной работы производственных систем.

Кроме того, логические контроллеры обладают широким набором функциональных возможностей, включая арифметические операции, обработку строковых данных, работу с аналоговыми сигналами, а также возможность работы с различными протоколами связи для обмена данными с другими устройствами в системе.

Эффективное использование основных принципов работы и функциональных возможностей логических контроллеров позволяет создавать гибкие, надежные и высокопроизводительные системы автоматизации, способные эффективно управлять различными производственными процессами и адаптироваться к изменяющимся требованиям и условиям.

Далее мы рассмотрим основные принципы работы и функциональные возможности логических контроллеров более детально, а также их применение в различных отраслях промышленности. Разбор конкретных типов логических контроллеров, их архитектуры и особенностей позволит лучше понять, как выбирать и применять подходящие решения для конкретных задач автоматизации. Также будет рассмотрено программирование и настройка логических контроллеров, включая различные языки программирования и методики, используемые в индустрии.

Дополнительно мы обратим внимание на последние тенденции в области промышленной автоматизации, включая развитие Интернета вещей (IoT), применение искусственного интеллекта (ИИ) и машинного обучения (МО) в системах управления, а также роль цифровой трансформации в промышленном секторе. Это позволит оценить перспективы развития и будущее использования логических контроллеров в промышленности.

Наконец, мы рассмотрим вопросы, связанные с обслуживанием и сопровождением систем автоматизации, включая диагностику и отладку, обновление программного обеспечения и аппаратных компонентов, а также управление изменениями и рисками. Это поможет обеспечить эффективную эксплуатацию и поддержку автоматизированных систем на всех этапах их жизненного цикла.

Затем мы сфокусируемся на применении логических контроллеров в различных отраслях промышленности. Рассмотрение конкретных примеров использования позволит лучше понять, как логические контроллеры применяются для автоматизации различных производственных процессов, начиная от простых задач управления до сложных систем мониторинга и управления.

Мы также обратим внимание на последние инновации в области логических контроллеров, такие как развитие облачных технологий и расширение возможностей связи через Интернет. Эти новые возможности предоставляют дополнительные инструменты для создания более гибких и эффективных систем автоматизации.

Программирование логических контроллеров также будет рассмотрено более детально, включая различные методики и языки программирования, используемые в индустрии. Это поможет разработчикам и инженерам выбирать наиболее подходящие инструменты для реализации конкретных задач.

Кроме того, мы рассмотрим вопросы безопасности и защиты данных в системах автоматизации на базе логических контроллеров. Это включает в себя обзор методов защиты от внешних угроз и внедрение современных систем мониторинга и аналитики для обеспечения безопасности и надежности работы системы.

В целом рассмотрение всех этих аспектов поможет создать полное представление о логических контроллерах и их роли в современной промышленной автоматизации, а также понять перспективы их развития в будущем.

Давайте начнем с более детального рассмотрения основных принципов работы и функциональных возможностей логических контроллеров. Это поможет нам лучше понять, как они управляются и какие задачи могут эффективно решать [4, 14, 18].

Разберем более подробно основные принципы работы и функциональные возможности логических контроллеров (PLC).

1. Обработка входных сигналов:

– логические контроллеры получают информацию от датчиков и других устройств в виде входных сигналов. Эти сигналы могут быть цифровыми (дискретными), такими как «включено/выключено», или аналоговыми (непрерывными), представляющими значения, такие как температура или давление;

– основная задача контроллера заключается в непрерывной обработке этих входных сигналов и принятии соответствующих решений в зависимости от их состояния.

2. Выполнение программы управления:

– PLC исполняет программу управления, которая определяет логику работы системы в зависимости от входных сигналов;

– программа может содержать различные логические операции, такие как условные операторы (IF-THEN-ELSE), циклы и таймеры, для реализации требуемого поведения системы.

Давайте представим простой пример программы для логического контроллера, который управляет работой светофора на перекрестке. В этой программе будут использоваться условные операторы (IF-THEN-ELSE), циклы и таймеры для реализации нужного поведения.

```pseudocode
START:

// Инициализация
SET Green_Light ON
SET Red_Light OFF
SET Timer_1 = 0

LOOP:

// Проверяем время работы зеленого сигнала
IF Timer_1 >= 30 THEN
    SET Green_Light OFF
    SET Yellow_Light ON
    WAIT 5 seconds
    SET Yellow_Light OFF
    SET Red_Light ON
    SET Timer_1 = 0
ELSE
    // Увеличиваем таймер работы зеленого сигнала
    INCREMENT Timer_1 by 1
END IF

WAIT 1 second

// Переходим к следующей итерации цикла
GO TO LOOP
```

В этом примере:

– светофор начинает работу с включенным зеленым сигналом и выключенным красным сигналом;

– затем программа начинает цикл, в котором каждую секунду проверяется таймер работы зеленого сигнала;

– если таймер достигает 30 с, то зеленый свет выключается и на 5 с включается желтый, после чего включается красный свет и таймер сбрасывается;

– программа затем возвращается к началу цикла и повторяет процесс.

Далее мы можем продолжить рассмотрение других аспектов работы логических контроллеров, таких как их архитектура, типы входных и выходных устройств, а также способы программирования и настройки.

1. Архитектура PLC:

– логические контроллеры имеют различную архитектуру, включая центральную (Centralized), децентрализованную (Distributed) и смешанную (Hybrid). Каждая архитектура имеет свои особенности и преимущества в зависимости от конкретного применения.

2. Типы входных и выходных устройств:

– PLC поддерживают широкий спектр входных и выходных устройств, включая дискретные (датчики, кнопки), аналоговые (термометры, датчики уровня) и специализированные устройства (энкодеры, приводы).

3. Способы программирования и настройки:

– существует несколько языков программирования для PLC, таких как логический рельсовый язык (Ladder Logic), структурный текст (Structured Text), функциональные блоки (Function Block Diagram), схема контактов (Contact Diagram) и последовательный блоковый код (Sequential Function Chart). Каждый из них имеет свои особенности и применяется в зависимости от задачи.

Давайте подробнее рассмотрим каждую архитектуру логических контроллеров.

1. Центральная (Centralized) архитектура:

– в центральной архитектуре все логические решения и задачи концентрируются в одном или нескольких центральных контроллерах, которые работают вместе как единое целое;

– эта архитектура обеспечивает простоту и удобство управления, так как весь процесс управления сосредоточен в одном месте;

– однако центральная архитектура может быть ограничена в масштабируемости и гибкости, особенно в случае больших и сложных систем.

2. Децентрализованная (Distributed) архитектура:

– в децентрализованной архитектуре логические контроллеры распределены по всей системе и выполняют свои задачи независимо друг от друга;

– это позволяет более гибко организовывать систему и обеспечивать высокую отказоустойчивость, так как отказ одного контроллера не приводит к полной остановке системы;

– однако децентрализованная архитектура требует более сложной организации и управления, так как каждый контроллер должен быть настроен и синхронизирован с остальными.

3. Смешанная (Hybrid) архитектура:

– смешанная архитектура комбинирует элементы центральной и децентрализованной архитектур, позволяя использовать преимущества обоих подходов в зависимости от конкретной задачи;

– например, в больших системах можно использовать центральный контроллер для общего управления и координации, а децентрализованные контроллеры для управления конкретными подсистемами или устройствами;

– это обеспечивает баланс между гибкостью и масштабируемостью системы.

Каждая из этих архитектур имеет свои преимущества и ограничения, и выбор конкретной архитектуры зависит от требований конкретного применения и условий эксплуатации.

Рассмотрение этих аспектов поможет более глубоко понять принципы работы логических контроллеров и их применение в различных промышленных областях.

4. Управление выходными сигналами:

– после обработки входных сигналов и выполнения программы управления PLC генерирует выходные сигналы для управления исполнительными механизмами, такими как моторы, клапаны, насосы и т. д.

Давайте представим пример формализации управления выходными сигналами с использованием псевдокода:

```pseudocode
IF Input_Signal = True THEN
    // Включаем выходной сигнал для управления мотором
    SET Motor_Output ON
ELSE
    // Выключаем выходной сигнал для управления мотором
    SET Motor_Output OFF
END IF

IF Pressure_Sensor > Threshold THEN
    // Открываем клапан для снижения давления
    SET Valve_Output OPEN
ELSE
```

```
    // Закрываем клапан
    SET Valve_Output CLOSED
END IF

IF Level_Sensor < Min_Level THEN
    // Включаем насос для подачи жидкости
    SET Pump_Output ON
ELSE IF Level_Sensor > Max_Level THEN
    // Отключаем насос, чтобы избежать перелива
    SET Pump_Output OFF
END IF
```

В этом примере:

– сначала проверяется состояние входного сигнала. Если он истинный, мы включаем выходной сигнал для управления мотором, в противном случае мы его выключаем;

– затем используются данные с датчика давления. Если давление превышает пороговое значение, мы открываем клапан для снижения давления. Если давление в пределах допустимого диапазона, мы закрываем клапан;

– наконец, проверяется уровень жидкости с помощью датчика уровня. Если уровень ниже минимального значения, мы включаем насос для подачи жидкости. Если уровень выше максимального значения, мы выключаем насос, чтобы предотвратить перелив.

Это простой пример, но он демонстрирует, как логический контроллер может управлять выходными сигналами в зависимости от состояния входных сигналов и заданных условий.

Давайте усложним пример, добавив возможность регулировки скорости двигателя в зависимости от значений сигналов.

```pseudocode
IF Input_Signal = True THEN
    // Включаем выходной сигнал для управления мотором
    SET Motor_Output ON

    // Регулируем скорость двигателя в зависимости от
значения сигнала скорости
    IF Speed_Signal = Low THEN
        SET Motor_Speed LOW_SPEED
    ELSE IF Speed_Signal = Medium THEN
        SET Motor_Speed MEDIUM_SPEED
    ELSE IF Speed_Signal = High THEN
        SET Motor_Speed HIGH_SPEED
    END IF
```

```
ELSE
    // Если входной сигнал ложный, выключаем мотор
    SET Motor_Output OFF
END IF

IF Pressure_Sensor > Threshold THEN
    // Открываем клапан для снижения давления
    SET Valve_Output OPEN
ELSE
    // Закрываем клапан
    SET Valve_Output CLOSED
END IF

IF Level_Sensor < Min_Level THEN
    // Включаем насос для подачи жидкости
    SET Pump_Output ON
ELSE IF Level_Sensor > Max_Level THEN
    // Отключаем насос, чтобы избежать перелива
    SET Pump_Output OFF
END IF
```

В этом усложненном примере:

– мы добавили переменную `Speed_Signal`, которая указывает на требуемую скорость двигателя в зависимости от значения сигнала скорости;

– при включении мотора мы проверяем значение сигнала скорости и соответствующим образом регулируем скорость двигателя;

– это позволяет динамически изменять скорость двигателя в зависимости от меняющихся условий работы системы, что может быть полезным для оптимизации процесса и энергопотребления;

– эти выходные сигналы могут быть цифровыми (для управления реле или транзисторами) или аналоговыми (для управления скоростью двигателей или уровнем сигнала).

5. Реальное время:

– важной характеристикой PLC является способность работать в режиме реального времени. Это означает, что контроллеры способны реагировать на изменения во входных сигналах и выполнять управляющие действия с минимальной задержкой.

Работа в режиме реального времени (Real-Time) является важной характеристикой логических контроллеров (PLC), особенно в промышленной автоматизации, где требуется точное и своевременное управление процессами и оборудованием. Давайте рассмотрим более подробно, что означает работа в режиме реального времени для PLC.

1. Точность и своевременность реакции. Работа в режиме реального времени означает, что контроллеры способны реагировать на входные сигналы и события с минимальной задержкой. Это важно для обеспечения точного управления и предотвращения нежелательных или аварийных ситуаций.

2. Особенности механизмов управления. Контроллеры PLC используют специальные механизмы управления, такие как прерывания (interrupts) и таймеры (timers), чтобы обеспечить реакцию в режиме реального времени. Прерывания позволяют контроллеру быстро переключаться на обработку важных событий, а таймеры используются для отслеживания временных задержек и периодических операций.

3. Оптимизация времени выполнения задач. Для обеспечения работы в режиме реального времени, разработчики программ PLC стремятся оптимизировать время выполнения задач, минимизируя задержки и избегая блокирующих операций, которые могут замедлить реакцию системы.

4. Приложения в промышленности. Работа в режиме реального времени особенно важна в промышленных приложениях, таких как управление производственными линиями, автоматические системы управления и мониторинга, где даже небольшие задержки могут иметь серьезные последствия для производственного процесса и безопасности.

5. Отличие от других систем: PLC отличаются от общих компьютерных систем, так как их аппаратная и программная архитектура специально разработана для обеспечения работы в режиме реального времени. Это означает, что PLC могут обрабатывать задачи с высокой степенью предсказуемости и точности, что важно для многих промышленных приложений.

Таким образом, работа в режиме реального времени является ключевой характеристикой PLC, обеспечивающей надежное и эффективное управление промышленными процессами и оборудованием.

Это особенно важно для промышленных приложений, где даже небольшие задержки могут иметь серьезные последствия.

Таким образом, логические контроллеры обеспечивают надежное и эффективное управление производственными процессами, позволяя создавать гибкие и адаптивные системы автоматизации.

Основные принципы работы логических контроллеров заключаются в том, что они обрабатывают входные сигналы, выполняют программу управления и управляют выходными сигналами.

Эти контроллеры выполняют операции в режиме реального времени, что позволяет им мгновенно реагировать на изменения в производственных процессах.

Функциональные возможности логических контроллеров включают в себя обработку различных типов входных сигналов, таких как цифровые и аналоговые сигналы, а также возможность работы с различными протоколами связи для обмена данными с другими устройствами в системе. Они также обладают арифметическими и логическими возможностями, что позволяет выполнять сложные вычисления и принимать решения на основе различных условий.

Эффективное использование этих принципов и возможностей логических контроллеров помогает создавать гибкие и надежные системы автоматизации, способные управлять различными производственными процессами и адаптироваться к изменяющимся условиям [3, 31, 36].

1.3. История и эволюция логических контроллеров

История и эволюция логических контроллеров представляет собой увлекательный путь развития технологий в области промышленной автоматизации. В начале своего пути логические контроллеры были разработаны как альтернатива реле и электромеханическим системам управления. Первые прототипы появились в середине XX в. и были созданы для автоматизации производственных процессов в промышленности.

В середине XX в., когда производственные процессы становились все более сложными и требовательными, промышленные предприятия столкнулись с необходимостью улучшить эффективность и надежность своих систем управления. На тот момент распространенными методами управления были реле и электромеханические системы. Однако они имели свои ограничения, такие как ограниченная гибкость и сложность настройки.

В этой обстановке появились первые прототипы логических контроллеров, которые представляли собой новый подход к автоматизации производственных процессов. Эти прототипы были разработаны как альтернатива электромеханическим системам управления и предлагали более гибкий и программируемый метод управления процессами.

Первые логические контроллеры были относительно простыми по сравнению с современными стандартами. Они состояли из базовых элементов, таких как логические блоки, таймеры и регистры, и могли выполнять ограниченный набор функций.

Первые логические контроллеры были построены на базе относительно простых элементов, которые позволяли осуществлять базовые операции управления и логической обработки данных. Ниже

приведены некоторые из основных элементов, которые составляли эти прототипы.

1. Логические блоки. Это основной строительный блок логических контроллеров. Логические блоки выполняли базовые логические операции, такие как AND, OR, NOT и т. д. Они использовались для сравнения входных сигналов, принятия решений и генерации выходных сигналов в зависимости от условий [12].

Ниже приведен пример кода на псевдокоде, демонстрирующий использование логических блоков для выполнения базовых логических операций:

```
// Пример программы на псевдокоде для управления выходным
сигналом на основе входных условий

// Определение входных сигналов
Input_Signal_A = True
Input_Signal_B = False

// Логический блок для выполнения операции AND
IF Input_Signal_A AND Input_Signal_B THEN
    // Если оба входных сигнала истинные, генерируем
выходной сигнал
    Output_Signal_AND = True
ELSE
    Output_Signal_AND = False
END IF

// Логический блок для выполнения операции OR
IF Input_Signal_A OR Input_Signal_B THEN
    // Если хотя бы один из входных сигналов истинный,
генерируем выходной сигнал
    Output_Signal_OR = True
ELSE
    Output_Signal_OR = False
END IF

// Логический блок для выполнения операции NOT
IF NOT Input_Signal_A THEN
    // Если входной сигнал ложный, генерируем выходной
сигнал
    Output_Signal_NOT = True
ELSE
    Output_Signal_NOT = False
END IF
```

```
// Вывод результатов
PRINT "Результат операции AND: " + Output_Signal_AND
PRINT "Результат операции OR: " + Output_Signal_OR
PRINT "Результат операции NOT: " + Output_Signal_NOT
```

В этом примере:

– у нас есть два входных сигнала, Input_Signal_A и Input_Signal_B, которые имеют различные значения (истина или ложь);

– мы используем логические блоки AND, OR и NOT для выполнения соответствующих операций над входными сигналами;

– в зависимости от результатов логических операций мы генерируем соответствующие выходные сигналы.

Давайте представим, что у нас есть промышленный процесс, в котором используется логический контроллер для управления работой конвейера в зависимости от различных входных условий. Допустим, у нас есть следующие входные данные.

1. Датчик присутствия продукта на конвейере (Product_Presence): истинный (True), если продукт присутствует на конвейере, ложный (False) — в противном случае.

2. Датчик перегрузки конвейера (Overload_Sensor): истинный, если конвейер перегружен, ложный — в противном случае.

3. Датчик аварийной остановки (Emergency_Stop): истинный, если сработала аварийная остановка, ложный — в противном случае.

Теперь давайте рассмотрим обработку этих входных данных с помощью логических операций на логическом контроллере.

```
// Входные данные
Product_Presence = True
Overload_Sensor = False
Emergency_Stop = False

// Логические операции
IF Product_Presence AND NOT Overload_Sensor AND NOT Emer-
gency_Stop THEN
    // Если продукт присутствует на конвейере, нет пере-
грузки и не сработала аварийная остановка,
    // то включаем конвейер
    Conveyor_Status = "Running"
ELSE
    // Иначе выключаем конвейер
    Conveyor_Status = "Stopped"
END IF
```

```
// Вывод результатов
PRINT "Статус конвейера: " + Conveyor_Status
```

В этом примере:

– мы используем логическую операцию AND для проверки нескольких условий одновременно;

– если продукт присутствует на конвейере, нет перегрузки и не сработала аварийная остановка, то мы включаем конвейер;

– если хотя бы одно из условий не выполняется, то мы останавливаем конвейер;

– результат операции выводится на экран;

– в конце программы результаты операций выводятся на экран.

2. Таймеры. Таймеры предоставляли возможность задержки выполнения определенных действий или изменения состояний сигналов. Они могли быть использованы для управления временными задержками в процессе или для выполнения периодических операций.

Ниже приведен пример использования таймера на псевдокоде для управления временными задержками в процессе.

```
// Задаем начальное состояние таймера и длительность
задержки
Timer_State = "Stopped"
Delay_Time = 5000 // Время задержки в миллисекундах
(например, 5 с)

// Основной цикл программы
LOOP:
    // Проверяем состояние таймера
    IF Timer_State == "Running" THEN
        // Если таймер работает, уменьшаем оставшееся
время на каждой итерации цикла
        Delay_Time = Delay_Time — Loop_Time //
Loop_Time — время одной итерации цикла
        IF Delay_Time <= 0 THEN
            // Если оставшееся время истекло,
останавливаем таймер
            Timer_State = "Stopped"
            // Выполняем действия после завершения
задержки
            Perform_Actions_After_Delay()
        END IF
    END IF

    // Продолжаем выполнение остальных операций программы
```

```
END LOOP

// Функция запуска таймера с указанной длительностью
задержки
FUNCTION Start_Timer(delay):
    Timer_State = "Running"
    Delay_Time = delay
END FUNCTION

// Функция выполнения действий после завершения задержки
FUNCTION Perform_Actions_After_Delay():
    // Выполняем необходимые действия
    PRINT "Действия после завершения задержки выполнены".
END FUNCTION
```

В этом примере:

– мы имеем переменную Delay_Time, которая представляет собой время задержки в миллисекундах;

– таймер начинает отсчет времени, когда его состояние устанавливается в Running;

– на каждой итерации основного цикла программа проверяет, истекло ли время задержки, и если да, то выполняет определенные действия;

– функция Start_Timer запускает таймер с указанной длительностью задержки;

– функция Perform_Actions_After_Delay определяет действия, которые будут выполнены после завершения задержки.

Ниже приведен пример реализации функции Perform_Actions_After_Delay на псевдокоде:

```
// Функция выполнения действий после завершения задержки
FUNCTION Perform_Actions_After_Delay():
    // Выполняем необходимые действия
    PRINT "Выполняются действия после завершения
задержки..."
    // Например, можем включить какое-то устройство или
выполнить определенную операцию
    // Здесь можно добавить любые другие необходимые
действия
END FUNCTION
```

Эта функция может быть расширена в соответствии с конкретными действиями, которые необходимо выполнить после завершения задержки. Например, вместо простого вывода сообщения о заверше-

нии задержки она может включать в себя более сложные действия, такие как управление исполнительными механизмами, обновление данных или отправка уведомлений.

Этот пример демонстрирует основные принципы использования таймеров для управления временными задержками в процессе выполнения программы логического контроллера.

3. Регистры. Регистры представляли собой небольшие участки памяти, в которых хранились временные данные, результаты логических операций или текущее состояние процесса. Они были важным средством для хранения и обработки информации внутри контроллера.

Регистры в логических контроллерах играют важную роль в хранении временных данных, результатах логических операций и текущего состояния процесса. Ниже более подробно объясняется их функциональность.

1. Хранение временных данных. Регистры предоставляют место для хранения временных данных, которые используются в процессе выполнения программы. Эти данные могут быть промежуточными результатами вычислений, значениями счетчиков или таймеров, а также другой информацией, которая временно требуется программе для корректной работы.

2. Хранение результатов логических операций. Регистры также могут использоваться для хранения результатов логических операций, таких как сравнения двух значений, проверка условий или выполнения других операций над булевыми данными. Например, результат сравнения двух чисел может быть сохранен в регистре для последующего использования в программе.

3. Отслеживание состояния процесса. Регистры могут служить для отслеживания текущего состояния процесса или устройства. Например, в промышленном процессе они могут использоваться для записи текущих значений датчиков, состояния исполнительных механизмов или других важных параметров, которые могут влиять на управление процессом.

4. Обработка информации. Регистры предоставляют средства для обработки информации внутри логического контроллера. Это может включать в себя выполнение арифметических операций, преобразование данных из одного формата в другой, а также другие манипуляции с данными, необходимые для корректной работы программы.

Использование регистров в логических контроллерах помогает эффективно управлять процессами, обеспечивая сохранение и обработку необходимой информации во время выполнения программы.

Они являются важным элементом в структуре данных контроллера, который позволяет эффективно реализовывать различные функции автоматизации и управления [18, 21].

Эти базовые элементы предоставляли логическим контроллерам возможность выполнять простые логические операции и управлять внешним оборудованием. Однако они были ограничены по своим возможностям и не могли обеспечить такой высокий уровень гибкости и функциональности, который стал доступен с развитием современных PLC. Тем не менее эти прототипы были первым шагом к созданию программируемых логических контроллеров и заложили основу для последующего развития в области промышленной автоматизации.

Однако они были революционными для своего времени, поскольку позволяли инженерам быстро перенастраивать системы управления без необходимости полной перекомпоновки электрической схемы.

Эти первые прототипы логических контроллеров стали первым шагом к программируемым логическим контроллерам (PLC), которые впоследствии стали неотъемлемой частью промышленной автоматизации. С развитием технологий их возможности и функциональность постоянно расширялись, и они продолжают играть ключевую роль в современной промышленности.

Один из ключевых моментов в истории логических контроллеров — внедрение программируемых логических контроллеров (PLC) в 1970-х гг. Это открыло новые возможности в автоматизации производства, позволяя быстро перенастраивать и программировать контроллеры для различных задач без необходимости полной перекомпоновки электрической схемы, как это было ранее с реле.

С течением времени технологии PLC продолжали развиваться, становясь все более мощными, компактными и гибкими. Внедрение микропроцессоров и цифровых технологий позволило значительно улучшить производительность и функциональность контроллеров. Сегодня существуют различные типы PLC — от простых моделей для небольших задач до высокопроизводительных систем, способных управлять сложными производственными процессами в режиме реального времени. Одним из современных трендов в эволюции логических контроллеров является интеграция сетевых технологий, таких как промышленный Ethernet и Интернет вещей (IoT), что позволяет создавать гибкие и распределенные системы автоматизации с возможностью удаленного мониторинга и управления.

Непрерывная эволюция технологий в области промышленной автоматизации продолжает расширять возможности и функциональ-

ность логических контроллеров, делая их неотъемлемой частью современного производства [23, 28].

1.4. Синтез программируемых логических схем

Синтез программируемых логических схем (ПЛС) представляет собой процесс автоматического создания логических схем на основе заданных функциональных требований. Этот процесс включает в себя несколько этапов, начиная с описания логики поведения схемы на некотором языке описания аппаратуры (например, языке Verilog или VHDL) и заканчивая с физической реализацией схемы на программируемом логическом устройстве (ПЛИ).

В начале процесса проектирования необходимо четко определить требования к функциональности логической схемы. Затем с использованием специализированных инструментов проектирования разработчики создают абстрактное описание схемы на одном из языков описания аппаратуры. Это описание включает в себя логические элементы, их связи и функции, которые должны быть реализованы.

После создания описания происходит синтез, который автоматически преобразует абстрактное описание схемы в набор логических элементов, таких как вентили, комбинационные и последовательные логические блоки. Этот процесс оптимизирует структуру схемы с целью минимизации затрат по ресурсам ПЛИ, таким как используемая площадь и время задержки.

Завершающий этап — физическая реализация схемы на программируемом логическом устройстве. На этом этапе сгенерированный код загружается в ПЛИ, где он программно настраивает внутреннюю структуру устройства в соответствии с описанием схемы. Полученное устройство готово к работе и выполняет функции, определенные в исходных требованиях.

Синтез программируемых логических схем играет ключевую роль в проектировании цифровых систем и устройств, позволяя быстро и эффективно создавать сложные логические схемы с минимальными затратами на разработку.

Давайте рассмотрим кейс применения синтеза программируемых логических схем в разработке системы управления светофором.

Кейс: разработка системы управления светофором.

Шаг 1: определение требований.

Первым шагом является определение требований к системе управления светофором, что включает определение длительности сигналов светофора, последовательности переключений между со-

стояниями, обработку сигналов датчиков, таких как датчики движения и пешеходные кнопки, и другие параметры.

Шаг 2: проектирование абстрактного описания.

На этом этапе разработчики создают абстрактное описание системы управления светофором на языке описания аппаратуры, таком как Verilog или VHDL. Описание включает логические блоки для управления светофорными сигналами, обработки входных сигналов датчиков и управления последовательностью переключений.

Шаг 3: синтез.

С использованием специализированных инструментов проектирования абстрактное описание системы светофора преобразуется в набор логических элементов, таких как комбинационные и последовательные блоки, необходимых для реализации заданных функций. Оптимизация происходит с учетом минимизации ресурсов ПЛИ и улучшения производительности.

Шаг 4: физическая реализация.

Сгенерированный код загружается в программируемое логическое устройство, которое программируется в соответствии с описанием схемы управления светофором. Полученное устройство способно автоматически управлять светофорными сигналами, обрабатывать входные сигналы от датчиков и обеспечивать правильную последовательность переключений в соответствии с заданными требованиями.

Конечно, давайте начнем процесс проектирования системы управления светофором с определения требований.

Шаг 1: определение требований.

1. Длительность сигналов светофора. Определить длительность каждого состояния светофора (красный, желтый, зеленый) в секундах.

2. Последовательность переключений. Установить последовательность переключений между состояниями светофора (например, красный → зеленый → желтый → красный).

3. Обработка сигналов датчиков. Разработать систему обработки сигналов от датчиков движения и пешеходных кнопок для управления переключением светофора.

4. Управление приоритетами. Реализовать функцию управления приоритетами для учета различных потоков транспорта и пешеходного движения.

5. Автоматический режим. Предусмотреть возможность перехода в автоматический режим работы светофора при отсутствии сигналов от датчиков.

6. Интерфейс управления. Создать пользовательский интерфейс для настройки параметров светофора и отображения текущего состояния.

Это основные требования, которые помогут определить функциональность и характеристики нашей системы управления светофором. На следующем этапе мы перейдем к проектированию абстрактного описания системы на языке описания аппаратуры (Verilog или VHDL).

Прежде чем мы приступим к написанию кода на языке Verilog, давайте определим основные модули и функциональность, которую мы должны реализовать.

1. Модуль управления светофором. Этот модуль будет отвечать за управление светофором, включая последовательность переключения сигналов и обработку входных сигналов датчиков.

2. Модуль таймера. Нам потребуется таймер для отслеживания длительности каждого состояния светофора.

3. Модуль датчиков. Этот модуль будет считывать сигналы от датчиков движения и пешеходных кнопок [24, 29].

4. Модуль приоритетов. Мы также можем реализовать модуль для управления приоритетами, чтобы обеспечить более эффективное управление светофором в зависимости от потоков транспорта и пешеходного движения.

Давайте начнем с написания основного модуля управления светофором.

```verilog
module TrafficLightController(
    input clk, // Входной тактовый сигнал
    input reset, // Входной сигнал сброса
    output reg red, // Сигнал красного света
    output reg yellow, // Сигнал желтого света
    output reg green // Сигнал зеленого света
);

// Параметры длительности состояний светофора (в тактах)
parameter RED_DURATION = 50 000; // Пример: 50 000 тактов
parameter YELLOW_DURATION = 10 000; // Пример: 10 000
тактов
parameter GREEN_DURATION = 60 000; // Пример: 60 000
тактов

// Перечисление состояний светофора
typedef enum logic [1:0] {
    RED_STATE,
```

```
    YELLOW_STATE,
    GREEN_STATE
} state_type;

// Регистр состояния светофора
reg [1:0] state;

// Счетчик для отслеживания длительности каждого
состояния
reg [15:0] counter;

// Процесс управления состоянием светофора
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Сброс состояния и счетчика при сигнале сброса
        state <= RED_STATE;
        counter <= 0;
    end else begin
        // Логика переключения состояний светофора
        case (state)
            RED_STATE: begin
                if (counter >= RED_DURATION) begin
                    state <= GREEN_STATE;
                    counter <= 0;
                end else begin
                    counter <= counter + 1;
                end
            end
            YELLOW_STATE: begin
                if (counter >= YELLOW_DURATION) begin
                    state <= RED_STATE;
                    counter <= 0;
                end else begin
                    counter <= counter + 1;
                end
            end
            GREEN_STATE: begin
                if (counter >= GREEN_DURATION) begin
                    state <= YELLOW_STATE;
                    counter <= 0;
                end else begin
                    counter <= counter + 1;
                end
            end
        endcase
    end
end
```

```
// Присваивание сигналов светофора в зависимости от
текущего состояния
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Сброс сигналов при сигнале сброса
        red <= 1'b0;
        yellow <= 1'b0;
        green <= 1'b0;
    end else begin
        case (state)
            RED_STATE: begin
                red <= 1'b1;
                yellow <= 1'b0;
                green <= 1'b0;
            end
            YELLOW_STATE: begin
                red <= 1'b0;
                yellow <= 1'b1;
                green <= 1'b0;
            end
            GREEN_STATE: begin
                red <= 1'b0;
                yellow <= 1'b0;
                green <= 1'b1;
            end
        endcase
    end
end

endmodule
```

Этот код представляет собой основной модуль управления светофором на языке Verilog. Он реализует логику переключения состояний светофора (красный, желтый, зеленый) и генерирует соответствующие сигналы для управления светодиодами или другими устройствами светофора [11, 18, 24].

Давайте подробно разберем этот код.

Мы создали модуль TrafficLightController, который представляет собой управление светофором. Этот модуль имеет несколько входных и выходных сигналов:

– clk: это тактовый сигнал, который определяет скорость работы системы;

– reset: сигнал сброса, который используется для восстановления начального состояния светофора;

– red, yellow, green: это выходные сигналы, которые управляют светофорными светодиодами.

Мы также определили параметры RED_DURATION, YELLOW_DURATION и GREEN_DURATION, которые определяют длительность каждого состояния светофора в тактах.

Внутри модуля есть процесс, который работает на каждом положительном фронте тактового сигнала или сигнала сброса. Этот процесс отслеживает текущее состояние светофора и использует счетчик для определения, когда переключиться на следующее состояние. Например, если текущее состояние — красный свет, и счетчик достиг длительности красного света, то мы переключаемся на зеленый свет и т. д.

Есть также другой процесс, который определяет, какие светодиоды должны быть включены в зависимости от текущего состояния светофора.

Например, если текущее состояние — красный свет, то только красный светодиод будет включен, а остальные выключены и т. д.

Давайте теперь переведем нашу логику управления светофором на уровень программной логики, используя базовые логические элементы, такие как вентили, триггеры и таймеры.

Для начала определим логику переключения светофора.

1. Логика переключения состояний. Нам нужно создать логику, которая будет переключать светофор между красным, желтым и зеленым состояниями в соответствии с заданной последовательностью и длительностью каждого состояния.

2. Логика управления светодиодами. Мы также должны определить логику, которая будет включать и выключать светодиоды в зависимости от текущего состояния светофора.

Для реализации этой логики нам понадобятся базовые логические элементы, такие как И, ИЛИ, НЕ, а также триггеры для отслеживания состояний и таймеры для подсчета времени каждого состояния.

Ниже приведен примерный набор логических элементов, которые могут быть использованы для реализации нашей системы управления светофором.

1. Логика переключения состояний:

– делитель частоты для создания временных задержек;

– счетчик для отслеживания времени в каждом состоянии;

– комбинационные логические элементы для определения условий переключения между состояниями.

2. Логика управления светодиодами:

– вентили для управления состоянием светодиодов в зависимости от текущего состояния светофора.

Далее будет предложена примерная схема, которая использует вышеупомянутые логические элементы для реализации управления светофором.

Таким образом, этот код моделирует работу простого светофора с помощью языка описания аппаратуры Verilog.

Этот кейс демонстрирует, как синтез программируемых логических схем может быть использован в разработке системы управления светофором для обеспечения эффективного и надежного функционирования системы светофора.

Начнем с разработки схемы управления светофором с использованием базовых логических элементов. Наша цель — создать логику, которая будет управлять переключением состояний светофора и управлением светодиодами для отображения соответствующих сигналов.

Прежде чем начнем, давайте определим состояния светофора и их длительность:

1) красный свет: обычно длительность около 60 с;

2) желтый свет: длительность около 5 с;

3) зеленый свет: обычно длительность около 60 с.

Теперь перейдем к разработке схемы управления светофором. Начнем с простой схемы, использующей базовые логические элементы, такие как вентили И, ИЛИ, НЕ и триггеры.

1. Счетчики и делители частоты. Можно использовать счетчики и делители частоты для отслеживания времени в каждом состоянии светофора.

2. Комбинационная логика. Нужно определить логику переключения между состояниями светофора в зависимости от текущего времени и входных сигналов.

3. Управление светодиодами. Будем использовать вентили для управления состоянием светодиодов в соответствии с текущим состоянием светофора.

Начнем с разработки базовой схемы управления светофором с использованием логических элементов.

Для начала определим, что может понадобиться.

1. Счетчики времени. Необходимо отслеживать время в каждом состоянии светофора.

2. Комбинационная логика. Для определения условий переключения между состояниями светофора.

3. Управление светодиодами. Для включения и выключения светодиодов в соответствии с текущим состоянием.

Начнем с создания простой схемы управления временем в каждом состоянии. Для этого можно использовать счетчики и делители частоты.

1. Счетчик красного состояния (Red State Counter). Этот счетчик будет отслеживать время в красном состоянии.

2. Счетчик желтого состояния (Yellow State Counter). Этот счетчик будет отслеживать время в желтом состоянии.

3. Счетчик зеленого состояния (Green State Counter). Этот счетчик будет отслеживать время в зеленом состоянии.

Для управления переходами между состояниями также может понадобиться комбинационная логика, которая будет определять условия переключения состояний в зависимости от текущего времени и входных сигналов.

Для определения комбинационной логики, которая будет определять условия переключения состояний светофора, можно использовать следующие правила.

1. Переход от красного к желтому свету происходит, когда истекло время красного света.

2. Переход от желтого к зеленому свету происходит, когда истекло время желтого света.

3. Переход от зеленого к красному свету происходит, когда истекло время зеленого света.

Также можно добавить дополнительные условия, такие как учет входных сигналов от дополнительных датчиков для определения необходимости переключения состояний вне зависимости от времени.

Ниже приведен пример комбинационной логики в псевдокоде.

```
if counter_red == N_red:
    // Переключение с красного на желтый свет
    set_yellow_light() // Включаем желтый светодиод
    clear_red_light() // Выключаем красный светодиод
else if counter_yellow == N_yellow:
    // Переключение с желтого на зеленый свет
    set_green_light() // Включаем зеленый светодиод
    clear_yellow_light() // Выключаем желтый светодиод
else if counter_green == N_green:
    // Переключение с зеленого на красный свет
    set_red_light() // Включаем красный светодиод

    clear_green_light() // Выключаем зеленый светодиод
```

Этот код определяет условия переключения состояний светофора в зависимости от текущего времени. В реальной реализации нужно учитывать входные сигналы от дополнительных датчиков, чтобы управление переключениями состояний было более гибким и адаптивным.

1.5. Методы диагностики и устранения неполадок

Методы диагностики и устранения неполадок в логических контроллерах являются важной частью обеспечения надежности и эффективности автоматизированных систем. В зависимости от конкретной проблемы и характеристик системы могут применяться различные методы и подходы.

Один из распространенных методов диагностики — это мониторинг состояния входных и выходных сигналов контроллера. Путем анализа входных сигналов и сравнения их с ожидаемыми значениями можно выявить отклонения и проблемы в работе системы.

Другой метод включает анализ программного кода контроллера на наличие ошибок или несоответствий, что может включать в себя проверку логики программы на наличие потенциальных ошибок или несоответствий требованиям [8, 26, 33].

Также важным методом является тестирование работы контроллера в различных режимах работы и с разными нагрузками. Проведение испытаний на производственном оборудовании или в условиях, максимально приближенных к реальным, позволяет выявить потенциальные проблемы и проверить работоспособность системы в разных сценариях.

Помимо этого, существуют специализированные инструменты и программное обеспечение для диагностики и мониторинга работы логических контроллеров. Они позволяют более подробно анализировать работу системы, выявлять проблемы и эффективно устранять неполадки.

Важно иметь в виду, что методы диагностики и устранения неполадок должны быть частью всей системы обеспечения качества и поддержки и систематически применяться для обеспечения стабильной и надежной работы автоматизированных процессов.

Конкретные примеры методов диагностики и устранения неполадок в логических контроллерах могут включать следующее.

1. Мониторинг входных и выходных сигналов. Например, если логический контроллер управляет системой освещения, его можно настроить на мониторинг входных сигналов от датчиков освещенно-

сти. Если значение сигнала выходит за пределы допустимого диапазона или не соответствует ожидаемому, это может указывать на проблему в сенсорах или в работе контроллера.

2. Анализ программного кода. Путем анализа программного кода логического контроллера можно выявить потенциальные ошибки или несоответствия требованиям. Например, если в коде обнаруживается условие, которое не учитывает определенную ситуацию или не обрабатывает определенные входные данные, это может привести к неправильной работе системы.

3. Тестирование работы контроллера. Проведение тестирования работы контроллера в различных режимах работы и с разными нагрузками позволяет выявить потенциальные проблемы. Например, при моделировании работы контроллера в среде симуляции можно проверить его поведение в разных сценариях и условиях работы.

4. Использование специализированных инструментов и программного обеспечения. Существуют специализированные инструменты и программное обеспечение для диагностики и мониторинга работы логических контроллеров. Например, программа для мониторинга состояния входных и выходных сигналов контроллера или для анализа его работы в режиме реального времени может помочь более эффективно выявить и устранить проблемы.

Эти методы могут быть использованы как по отдельности, так и в комбинации для обеспечения надежной работы логических контроллеров и своевременного устранения выявленных проблем.

ГЛАВА 2. ЯЗЫКИ ПРОГРАММИРОВАНИЯ PLC

2.1. Ladder Diagram (LD): синтаксис и примеры

Язык программирования Ladder Diagram (LD) представляет собой графический язык, который используется для программирования логических контроллеров. Синтаксис этого языка основан на принципах электрических схем, что делает его интуитивно понятным для инженеров и техников, занимающихся автоматизацией.

В Ladder Diagram программа представляется в виде схемы, состоящей из вертикальных столбцов (рельсов), на которых размещаются контакты и катушки. Каждый столбец представляет собой состояние входов и выходов в определенный момент времени.

Контакты в Ladder Diagram могут представлять как физические входные сигналы, так и промежуточные логические условия. Они могут быть соединены между собой с помощью различных логических операторов, таких как AND, OR, NOT, чтобы формировать сложные условия и логические цепочки.

Пример программы на языке Ladder Diagram для управления двигателем может выглядеть следующим образом:

```
|----[ ]----[ ]----( )----|
|   Start   Stop   Motor  |
|-------------------------|
```

В этом примере при условии, что физические кнопки Start и Stop нажаты, контакты Start и Stop замкнуты. Это приведет к замыканию контакта Motor, что, в свою очередь, запустит двигатель. Как только кнопка Stop будет отпущена, контакт Stop разомкнется, останавливая двигатель.

Ladder Diagram является одним из наиболее распространенных языков программирования для логических контроллеров благодаря своей простоте и интуитивной понятности. Он удобен для разработки программ управления, особенно в области промышленной автоматизации.

Ключевой чертой Ladder Diagram является его интуитивно понятный характер, который делает его привлекательным для инженеров и техников, не имеющих глубоких знаний в программировании. Важно отметить, что в языке Ladder Diagram нет строгой структуры программирования, как в текстовых языках программирования. Вместо этого программа представляется в виде графической схемы, что делает ее легко визуализируемой и отлаживаемой.

Программа на языке Ladder Diagram представляется в виде графической схемы, состоящей из различных элементов, таких как контакты, катушки, реле и таймеры, которые располагаются на вертикальных столбцах, называемых рельсами. Каждый столбец представляет определенный момент времени, а состояние входных и выходных устройств отображается в этот момент времени.

Контакты в Ladder Diagram могут быть как физическими входными и выходными устройствами, так и виртуальными логическими элементами, которые используются для создания условий и логических связей в программе. Эти контакты могут быть соединены между собой с помощью различных логических операторов, таких как AND, OR, NOT, что позволяет создавать сложные логические цепочки.

Давайте рассмотрим пример программы на языке Ladder Diagram для управления системой освещения с использованием физических кнопок «Включить» и «Выключить» и лампочки «Лампа».

```
|----[ ]------( )------|
|  Включить  Лампа    |
|----[ ]------( )------|
|  Выключить Лампа    |
|----------------------|
```

В этой программе имеются два контакта, соответствующих кнопкам «Включить» и «Выключить», и два контакта, управляющих состоянием лампочки «Лампа». Когда кнопка «Включить» нажата, контакт «Включить» замыкается, что приводит к замыканию контакта «Лампа» и включению лампочки. При этом если кнопка «Выключить» нажата, то контакт «Выключить» замыкается, что приводит к размыканию контакта «Лампа» и выключению лампочки.

Этот пример демонстрирует использование языка Ladder Diagram для простого управления системой освещения. Графическая нотация Ladder Diagram делает программу легко визуализируемой и понятной, что облегчает ее разработку и отладку.

Давайте усложним пример и добавим возможность управления несколькими лампочками с использованием дополнительной логики.

```
|----[ ]------[ ]------( )------|
|  Включить  Лампа1  Лампа2  |
|----[ ]------[ ]------( )------|
|  Выключить Лампа1  Лампа2  |
|-----------------------------|
```

В этом примере у нас есть две кнопки «Включить» и «Выключить» и две лампочки «Лампа1» и «Лампа2». Каждая кнопка управляет соответствующим контактом, а контакты управляют состоянием соответствующих лампочек. При нажатии на кнопку «Включить» соответствующая лампочка включается, а при нажатии на кнопку «Выключить» — выключается.

Теперь давайте добавим логику, чтобы обеспечить включение обеих лампочек только в том случае, если обе кнопки «Включить» нажаты.

```plaintext
|----[ ]------[ ]------( )------|
|  Включить  Включить  Лампа1   |
|----[ ]------[ ]------( )------|
|  Выключить Лампа1   Лампа2   |
|-----------------------------|
```

Таким образом, в этой программе обе лампочки будут включены только в том случае, если обе кнопки «Включить» нажаты. Если одна из кнопок «Выключить» нажата, то соответствующая лампочка выключится, но другая останется включенной, если кнопка «Включить» остается нажатой.

Этот пример демонстрирует более сложную логику управления системой освещения с использованием языка Ladder Diagram. Графическое представление делает программу легко воспринимаемой и управляемой, что особенно важно в промышленной автоматизации.

Программирование на Ladder Diagram основано на принципе выполнения последовательности действий, которые представлены на графической схеме. При этом каждое действие происходит в определенном порядке, начиная с верхних рельсов и двигаясь вниз. Это делает программу легко визуализируемой и понятной для техников и инженеров, работающих с системой.

Для отладки программы на Ladder Diagram обычно используются специализированные инструменты, которые позволяют выполнять симуляцию работы программы и анализировать ее поведение в различных сценариях. Такие инструменты позволяют легко выявлять и исправлять ошибки в программе до ее загрузки на реальный логический контроллер, что повышает эффективность и надежность работы системы.

Программы на языке Ladder Diagram обычно разрабатываются с использованием специализированных инструментов, таких как программное обеспечение для программирования логических контроллеров. Эти инструменты предоставляют широкий набор графических

элементов, которые можно использовать для создания схемы, а также возможности симуляции и отладки программы перед ее загрузкой на реальный контроллер.

Программирование позволяет создавать надежные и эффективные программы управления, особенно в области промышленной автоматизации, где важными являются простота, надежность и прозрачность работы программы [123–124]. Однако, несмотря на свою популярность, он может иметь некоторые ограничения в сложных сценариях программирования, где требуются более сложные логические операции или алгоритмы управления.

В целом Ladder Diagram остается одним из основных языков программирования для логических контроллеров благодаря своей простоте, надежности и широкому распространению в промышленности.

Давайте рассмотрим промышленный кейс использования логических контроллеров для управления системой конвейера на производственной линии.

Предположим, у нас есть производственная линия, на которой производится сборка различных изделий. Эта линия состоит из нескольких станций, включая станцию подачи материалов, станцию сборки, станцию упаковки и т. д. Каждая станция оборудована конвейером для перемещения изделий между ними.

Наша задача — разработать систему управления для этой линии с помощью логических контроллеров. Опишем основные этапы этого процесса.

1. Сначала определим требования к системе управления, включая логику работы каждой станции, последовательность перемещения изделий по линии, автоматическое обнаружение и устранение неполадок и т. д.

2. На основе требований спроектируем систему управления, включая распределение логических контроллеров по станциям, определение входных и выходных сигналов, разработку логики управления для каждой станции и между ними, а также создание средств диагностики и мониторинга.

3. Для каждого логического контроллера разработаем программу на языке Ladder Diagram или другом подходящем языке программирования PLC. Эта программа включает логику работы конвейера, включая управление скоростью движения, обработку сигналов с датчиков, управление приводами и т. д.

4. После разработки программа тестируется и отлаживается на симуляторе или в реальных условиях производства. В ходе тестирова-

ния проверяется правильность работы системы управления, ее надежность, а также обнаруживаются и устраняются возможные ошибки.

5. После успешного завершения тестирования система управления внедряется на производственной линии. Проводится обучение персонала, осуществляется техническая поддержка и обновление системы в соответствии с потребностями производства.

Таким образом, использование логических контроллеров позволяет эффективно управлять сложными производственными процессами, обеспечивая их автоматизацию, надежность и высокую производительность.

Давайте приступим к разработке программы управления для системы конвейера на производственной линии с использованием языка Ladder Diagram.

Для начала определим основные функциональные блоки нашей системы.

1. Станция подачи материалов. Эта станция отвечает за загрузку сырья на конвейер.

2. Станция сборки. Здесь происходит сборка изделий из загруженного сырья.

3. Станция упаковки. После сборки изделия упаковываются для отправки.

4. Система конвейера. Осуществляет перемещение изделий между станциями.

Теперь разработаем основную логику работы конвейера.

1. При запуске системы конвейер останавливается.

2. После нажатия кнопки «Старт» конвейер начинает движение.

3. При достижении каждой станции конвейер останавливается для выполнения соответствующих операций.

4. По завершении операций конвейер продолжает движение к следующей станции.

5. Процесс повторяется до завершения всех операций на линии.

6. После завершения работы всех станций конвейер останавливается.

Давайте начнем с разработки программы на языке Ladder Diagram для управления этой логикой. Начнем с описания логики работы станции подачи материалов.

Для описания логики работы станции подачи материалов на конвейере воспользуемся языком Ladder Diagram (LD). Наша задача состоит в том, чтобы контролировать процесс загрузки сырья на конвейер и управлять его движением.

Для начала определим основные элементы и условия, которые необходимо учитывать.

1. Датчик загрузки. Датчик, который определяет наличие сырья для загрузки на конвейер.

2. Мотор конвейера. Мотор, который обеспечивает движение конвейера.

3. Кнопка «Старт». Кнопка, которая запускает процесс работы конвейера.

Теперь давайте опишем логику работы станции подачи материалов в виде программы на языке Ladder Diagram:

```
|-------------------------|
|  ---[ ]---( )-----------|     |-----------|
| | Датчик  |               |   |    Мотор    |
| | загрузки| ----( )----|     |------------|
|  ---[ ]---( )-----------|     |------------|
|          Кнопка "Старт"           |
|-------------------------|
```

Объяснение:

– датчик загрузки представлен контактом, который замыкается, когда на конвейере появляется сырье для загрузки;

– кнопка «Старт» представлена контактом, который замыкается при нажатии на кнопку запуска;

– мотор конвейера представлен катушкой (выходным устройством), которая включается, когда условия для загрузки сырья и нажатия кнопки «Старт» выполнены.

Это основная логика работы станции подачи материалов. Когда сырье обнаружено и нажата кнопка «Старт», мотор конвейера включается, начиная движение конвейера.

Дополним нашу программу управления конвейером для станции подачи материалов, чтобы обеспечить более надежное и гибкое управление. Для этого добавим логику автоматической остановки конвейера при достижении определенных условий.

Рассмотрим следующие уточнения и дополнения.

1. Остановка при достижении конца линии. Когда сырье загружено на конвейер и кнопка «Старт» нажата, конвейер должен двигаться до тех пор, пока не достигнет конца линии, где будет расположен датчик конца конвейера.

2. Автоматическая остановка. После достижения конца линии конвейер должен автоматически остановиться для предотвращения столкновения с изделиями или препятствиями.

Давайте добавим эту логику к нашей программе:

```plaintext
|-------------------------|
|  ---[ ]---( )-----------|     |-----------|
| | Датчик  |                |   |    Мотор    |
| | загрузки| ----( )----|     |-----------|
|  ---[ ]---( )-----------|     |-----------|
|          Кнопка "Старт"   |   |-----------|
|-------------------------|     | Датчик конца |
                                  | конвейера |
                                  |---[ ]------|
```

Объяснение:

– датчик конца конвейера представлен контактом, который замыкается, когда конвейер достигает конца линии;

– при замыкании контакта датчика конца конвейера мотор конвейера отключается, что приводит к его остановке.

Теперь у нас есть более полная программа управления для станции подачи материалов, которая обеспечивает безопасное и эффективное движение конвейера, учитывая условия загрузки и конца линии.

2.2. Function Block Diagram (FBD): основы и примеры

Function Block Diagram (FBD) — это еще один из языков программирования, используемых в логических контроллерах. Он представляет собой графический язык, который позволяет создавать программы, используя функциональные блоки и их взаимодействие.

Основные элементы FBD включают в себя функциональные блоки, входные и выходные контакты, а также логические соединения между ними. Каждый функциональный блок выполняет определенную функцию, а взаимодействие между ними определяет логику программы.

Пример программы на FBD для управления конвейером может выглядеть следующим образом:

```
        +-------------+
    +---| Load Sensor |---+
    |   +-------------+   |
    |                     |
    |   +-------------+   |
    +---| Start Button|---+
    |   +-------------+   |
    |                     |
```

```
    |    +--------------+   |
    +----| Conveyor Motor|---+
         +--------------+
```

В этом примере:

– Load Sensor — функциональный блок, представляющий датчик загрузки;

– Start Button — функциональный блок, представляющий кнопку запуска;

– Conveyor Motor — функциональный блок, представляющий мотор конвейера.

Соединение между блоками показывает поток данных или управления. Например, выход датчика загрузки может быть подключен ко входу мотора конвейера, чтобы запустить его при обнаружении загруженного материала.

Function Block Diagram обеспечивает наглядное представление логики программы, что упрощает ее понимание и отладку.

Примеры использования Function Block Diagram (FBD) включают создание программ управления для различных промышленных процессов и систем автоматизации. Ниже приводятся несколько конкретных примеров.

1. Управление конвейерной линией. Программа на FBD может управлять работой конвейера, регулируя его скорость, направление движения и остановку в зависимости от условий загрузки и других факторов.

2. Управление системой отопления и вентиляции. FBD может использоваться для создания программы управления системой отопления и вентиляции в здании. Это включает в себя контроль работы котла, насосов, клапанов и вентиляторов для поддержания комфортной температуры и вентиляции в помещении.

3. Управление промышленным роботом. Программа на FBD может управлять действиями промышленного робота на производственной линии. Это включает в себя управление движениями робота, его захватом и выпуском предметов, а также координацию его действий с другими устройствами на линии.

4. Управление системой освещения и энергоснабжения: FBD может использоваться для создания программы управления системой освещения и энергоснабжения в здании или на территории предприятия. Это включает в себя контроль работы светильников, выключателей, генераторов и других устройств для оптимизации энергопотребления и обеспечения безопасности.

Эти примеры демонстрируют широкий спектр применения Function Block Diagram в различных областях промышленности и автоматизации, где требуется надежное и эффективное программное управление системами и процессами.

2.3. Structured Text (ST): структура и особенности

Structured Text (ST) — это текстовый язык программирования, используемый в логических контроллерах для создания программ управления. Он имеет структурированный синтаксис, основанный на алгоритмических конструкциях, что делает его ближе к стандартным языкам программирования, таким как Pascal или C.

Основные особенности Structured Text:

– простота чтения и написания: ST использует простой и понятный синтаксис, что делает его легко читаемым и понятным для разработчиков;

– мощные операторы и функции: в ST доступны различные операторы и функции для выполнения различных задач, включая математические операции, логические операции, обработку строк и т. д.;

– структурированные конструкции: ST поддерживает структурированные конструкции программирования, такие как циклы, условные операторы, функции и процедуры, что позволяет организовывать код и повышать его читаемость и модульность.

Структура программы на Structured Text состоит из последовательности инструкций, которые выполняются поочередно. Программа может содержать объявления переменных, определения функций и процедур, а также основной код, который определяет логику управления системой.

Structured Text широко используется в промышленной автоматизации для создания сложных программ управления, так как он обеспечивает высокую гибкость и возможность реализации различных алгоритмов и логики работы систем.

Далее можно рассмотреть пример структуры программы на Structured Text и ее основные элементы.

Программа на Structured Text обычно состоит из следующих элементов.

1. Объявление переменных. Этот раздел содержит объявления всех переменных, которые будут использоваться в программе. Включает в себя как входные, так и выходные переменные, а также промежуточные переменные для хранения промежуточных результатов.

2. Определение функций и процедур. В этом разделе можно определить пользовательские функции и процедуры, которые будут использоваться в программе. Это позволяет организовать код программы в более модульную структуру и повторно использовать его части.

3. Основной код программы. Этот раздел содержит основной код программы, включающий последовательность инструкций, которые выполняются поочередно. Здесь реализуется логика управления системой, включая условные операторы, циклы, математические операции и другие действия.

Пример программы на Structured Text для управления конвейером может выглядеть следующим образом:

```plaintext
VAR
    StartButton: BOOL;     ( Входной сигнал от кнопки
запуска )
    LoadSensor: BOOL;       ( Входной сигнал от датчика
загрузки )
    ConveyorMotor: BOOL;    ( Выходной сигнал для
управления мотором конвейера )

( Основной код программы )
BEGIN
    ( Если нажата кнопка запуска и обнаружено сырье )
    IF StartButton AND LoadSensor THEN
        ConveyorMotor := TRUE;  ( Включить мотор
конвейера )
    ELSE
        ConveyorMotor := FALSE; ( Выключить мотор
конвейера )
    END_IF;
END;
```

Это пример простой программы на Structured Text, которая управляет мотором конвейера в зависимости от состояния кнопки запуска и датчика загрузки. Программа использует структурированные конструкции IF...THEN...ELSE для принятия решений и управления выходным сигналом для мотора конвейера.

Продолжим рассмотрение особенностей языка Structured Text (ST) и его применения.

4. Циклы и условные операторы. В ST доступны различные циклы (например, циклы FOR, WHILE) и условные операторы

(IF...THEN...ELSE), которые позволяют создавать сложную логику управления, основанную на условиях и повторяющихся действиях.

5. Функции и процедуры. ST поддерживает определение пользовательских функций и процедур, которые могут быть использованы для упрощения кода, его модульности и повторного использования.

6. Математические и логические операции. В ST доступны различные математические операции (сложение, вычитание, умножение и деление) и логические операции (AND, OR, NOT), которые позволяют выполнять различные вычисления и обработку данных.

Программы на Structured Text используются для управления различными системами и процессами в промышленной автоматизации, такими как производственные линии, системы отопления и вентиляции, системы освещения, системы контроля доступа и многие другие. Этот язык программирования обладает высокой гибкостью и универсальностью, что делает его широко применимым в различных отраслях промышленности и инженерии.

Продолжая обсуждение языка программирования Structured Text (ST), важно отметить его основные преимущества и возможности.

7. Структурная ясность. ST обеспечивает структурную ясность кода благодаря использованию четких алгоритмических конструкций, что упрощает понимание логики программы.

8. Высокая производительность. Программы, написанные на ST, обычно имеют высокую производительность, поскольку могут быть эффективно скомпилированы и выполнены на логическом контроллере.

9. Поддержка различных типов данных. ST поддерживает различные типы данных, такие как целые числа, вещественные числа, булевы значения, строки и т. д., что позволяет разработчикам работать с различными типами информации.

10. Гибкость и расширяемость. Благодаря своей гибкости и возможности использования различных алгоритмических конструкций ST обеспечивает высокую степень гибкости и расширяемости для создания сложных программ управления.

Программы на Structured Text могут быть разработаны с использованием различных интегрированных сред разработки (IDE), предоставляемых производителями логических контроллеров, таких как Siemens, Allen-Bradley, Schneider Electric и др. Эти среды обеспечивают возможность отладки, моделирования и тестирования программ непосредственно на целевом оборудовании или в виртуальной среде, что обеспечивает эффективное развертывание и поддержку промышленных автоматизированных систем.

Ниже приведен пример простой программы на языке Structured Text для управления системой освещения в здании.

```plaintext
PROGRAM LightingControl
VAR
    LightSensor: BOOL;       ( Входной сигнал от датчика
освещенности )
    ManualSwitch: BOOL;      ( Входной сигнал от
переключателя освещения )
    LightOutput: BOOL;       ( Выходной сигнал для
управления освещением )

( Основной код программы )
BEGIN
    ( Если датчик освещенности обнаруживает недостаточное
освещение и переключатель включен )
    IF LightSensor AND ManualSwitch THEN
        LightOutput := TRUE;    ( Включить освещение )
    ELSE
        LightOutput := FALSE;   ( Выключить освещение )
    END_IF;
END_PROGRAM
```

Этот пример программы управляет освещением в помещении в зависимости от двух условий: уровня освещенности, обнаруженного датчиком, и положения переключателя освещения. Если уровень освещенности ниже заданного порога и переключатель включен, то программа включает освещение, иначе оно выключается.

Давайте усложним программу, добавив возможность управления яркостью освещения с помощью регулятора. Это может выглядеть так:

```plaintext
PROGRAM LightingControl
VAR
    LightSensor: BOOL;          ( Входной сигнал от
датчика освещенности )
    ManualSwitch: BOOL;         ( Входной сигнал от
переключателя освещения )
    DimmerSwitch: INT := 0;     ( Входной сигнал от регу-
лятора яркости (0 — выключен, 100 — максимальная яркость)
)
    LightOutput: REAL;          ( Выходной сигнал для
управления яркостью освещения )

( Основной код программы )
BEGIN
```

```
    ( Если датчик освещенности обнаруживает недостаточное
освещение и переключатель включен )
    IF LightSensor AND ManualSwitch THEN
        LightOutput := 100;     ( Установить максимальную
яркость )
    ELSE
        LightOutput := 0;       ( Выключить освещение )
    END_IF;
    ( Управление яркостью освещения с помощью регулятора
)
    IF DimmerSwitch > 0 THEN
        LightOutput := LightOutput  DimmerSwitch / 100;
( Установить яркость в соответствии с положением
регулятора )
    END_IF;
END_PROGRAM
```

В этом обновленном примере добавлен регулятор яркости DimmerSwitch, который позволяет пользователю регулировать яркость освещения от 0 до 100%. Если переключатель включен и датчик освещенности обнаруживает недостаточное освещение, программа устанавливает максимальную яркость. Затем, если регулятор яркости установлен на значение больше 0, программа устанавливает яркость освещения в соответствии с положением регулятора.

Давайте еще усложним программу, добавив функцию автоматического выключения освещения через определенное время после того, как переключатель выключен. Это можно сделать так:

```plaintext
PROGRAM LightingControl
VAR
    LightSensor: BOOL;              ( Входной сигнал от
датчика освещенности )
    ManualSwitch: BOOL;             ( Входной сигнал от
переключателя освещения )
    DimmerSwitch: INT := 0;         ( Входной сигнал от ре-
гулятора яркости (0 — выключен, 100 — максимальная яр-
кость) )
    Timer: TIME := T0s;           ( Таймер для
автоматического выключения освещения )
    LightOutput: REAL;              ( Выходной сигнал для
управления яркостью освещения )

( Основной код программы )
BEGIN
```

```
    ( Если датчик освещенности обнаруживает недостаточное
освещение и переключатель включен )
    IF LightSensor AND ManualSwitch THEN
        LightOutput := 100;        ( Установить
максимальную яркость )
        Timer := T0s;              ( Сбросить таймер
выключения )
    ELSIF NOT ManualSwitch THEN
        ( Если переключатель выключен, уменьшать яркость
освещения )
        IF Timer <= T10m THEN    ( Если прошло менее 10
мин после выключения переключателя )
            LightOutput := LightOutput  (10 —
TIME_TO_INT(Timer)) / 10;    ( Уменьшить яркость
пропорционально времени )
        ELSE
            LightOutput := 0;     ( Полностью выключить
освещение )
        END_IF;
    END_IF;

    ( Управление яркостью освещения с помощью регулятора
)
    IF DimmerSwitch > 0 THEN
        LightOutput := LightOutput  DimmerSwitch / 100;
( Установить яркость в соответствии с положением
регулятора )
    END_IF;
END_PROGRAM
```

В этой версии программы добавлен таймер Timer, который запускается при выключении переключателя. Если переключатель выключен, программа постепенно уменьшает яркость освещения в течение 10 мин, после чего полностью выключает освещение. Если переключатель снова включается до истечения времени таймера, то освещение возвращается к максимальной яркости и таймер сбрасывается.

Продолжим рассмотрение примеров кейсов, которые могут быть реализованы с помощью программирования логических контроллеров.

1. Управление системой подачи воды. Программа может контролировать работу насосов и клапанов для обеспечения надежной подачи воды в систему водоснабжения.

2. Управление системой кондиционирования воздуха в здании. Программа может контролировать работу кондиционеров и вентиля-

торов для поддержания комфортной температуры и влажности в помещениях.

3. Управление системой освещения на автомобильном заводе. Программа может автоматически включать и выключать светильники на производственном участке в зависимости от наличия сотрудников и стадии производства.

4. Управление системой энергосбережения в здании. Программа может оптимизировать использование энергии, автоматически выключая освещение и системы кондиционирования воздуха в пустых помещениях.

5. Управление системой подачи газа в промышленном оборудовании. Программа может контролировать работу клапанов и регуляторов давления для обеспечения безопасной и эффективной подачи газа в производственное оборудование.

Эти примеры демонстрируют широкий спектр возможностей использования программирования логических контроллеров для автоматизации различных процессов в промышленности, строительстве, сельском хозяйстве и других областях.

2.4. Instruction List (IL): описание и примеры использования

Instruction List (IL) — это текстовый язык программирования, который используется для написания программ для логических контроллеров. IL представляет собой набор инструкций, каждая из которых выполняет определенное действие или операцию. В отличие от графических языков, таких как Ladder Diagram (LD) и Function Block Diagram (FBD), IL позволяет представить программу в текстовой форме, что делает ее более компактной и удобной для написания.

Пример IL-кода для управления освещением может выглядеть так:

```
NETWORK 1:
    LD LightSensor AND ManualSwitch        ( Если датчик
освещенности и переключатель включены )
    OUT LightOutput := 1;                  ( Включить
освещение )
END_NETWORK

NETWORK 2:
    NOT ManualSwitch                       ( Если
переключатель выключен )
    AND TimerDone                          ( И таймер
истек )
```

```
    OUT LightOutput := 0;                    ( Выключить
освещение )
END_NETWORK
```

В этом примере первая сеть (NETWORK 1) включает освещение, если датчик освещенности и переключатель включены. Вторая сеть (NETWORK 2) выключает освещение, если переключатель выключен и таймер завершен. Каждая инструкция в IL выполняет определенное действие, такое как логическая операция AND или присваивание значения выходу.

Давайте рассмотрим еще один пример использования Instruction List (IL) для управления системой автоматического полива растений. В этом примере мы будем использовать IL для создания программы, которая будет контролировать работу насоса для подачи воды в систему полива на основе данных о влажности почвы.

```
NETWORK 1:
    LD SoilMoistureSensor > Threshold    ( Если влажность
почвы ниже порогового значения )
    AND NOT PumpRunning                  ( И насос не
работает )
    OUT StartPump := 1;                  ( Запустить
насос )
END_NETWORK

NETWORK 2:
    LD SoilMoistureSensor <= Threshold   ( Если влажность
почвы достигла порогового значения )
    OR PumpRunning                       ( Или насос уже
работает )
    OUT StopPump := 1;                   ( Остановить
насос )
END_NETWORK
```

В этом примере первая сеть (NETWORK 1) включает насос, если влажность почвы ниже заданного порогового значения и насос не работает. Вторая сеть (NETWORK 2) останавливает насос, если влажность почвы достигла порогового значения или насос уже работает.

IL позволяет легко выразить логику управления в текстовой форме, что делает программу более компактной и удобной для написания и отладки.

Синтаксис Instruction List (IL) — это формат представления инструкций программы для логического контроллера в текстовой фор-

ме. В IL каждая строка представляет собой отдельную инструкцию, которая выполняет определенное действие. Рассмотрим основные аспекты синтаксиса IL более подробно.

1. Инструкции и сети. Программа на IL состоит из набора сетей, которые, в свою очередь, содержат инструкции. Каждая сеть начинается с ключевого слова NETWORK и завершается ключевым словом END_NETWORK.

2. Инструкции. Инструкции в IL выполняют определенные действия, такие как чтение или запись значения входного или выходного сигнала, выполнение логических операций, управление таймерами и т. д. Каждая инструкция представляет собой отдельную строку кода и завершается точкой с запятой (;).

3. Логические операции. IL поддерживает стандартные логические операции, такие как AND, OR, NOT, а также операции сравнения (>, <, = и т. д.). Операнды операций обычно представляют собой входные или внутренние переменные.

4. Условные инструкции. IL поддерживает условные инструкции, такие как IF...THEN...ELSE..., которые позволяют выполнять различные действия в зависимости от условий.

5. Комментарии. В IL можно добавлять комментарии, которые начинаются с символа «(» и заканчиваются символом «)». Комментарии используются для пояснения кода и делают его более понятным для других разработчиков.

Пример IL-кода для управления освещением, который рассматривался ранее, выглядит следующим образом:

```
NETWORK 1:
    LD LightSensor AND ManualSwitch        ( Если датчик
освещенности и переключатель включены )
    OUT LightOutput := 1;                  ( Включить
освещение )
END_NETWORK

NETWORK 2:
    NOT ManualSwitch                       ( Если
переключатель выключен )
    AND TimerDone                          ( И таймер
истек )
    OUT LightOutput := 0;                  ( Выключить
освещение )
END_NETWORK
```

В этом примере LD, AND, NOT, OUT — это ключевые слова, представляющие инструкции, которые выполняют логические операции и управляют выходными сигналами. Комментарии помогают понять, какая логика реализована в каждой сети.

Дополним описание синтаксиса IL еще некоторыми важными аспектами.

6. Входные и выходные переменные. IL позволяет использовать входные и выходные переменные, которые представляют собой состояния внешних устройств или результаты выполнения программы. Входные переменные обычно представляются как контакты, а выходные — как катушки. Их состояние может быть изменено выполнением соответствующих инструкций.

7. Временные задержки и таймеры. IL поддерживает инструкции для управления временными задержками и таймерами. Это позволяет программе ожидать определенное время перед выполнением следующих действий или переключением состояний.

8. Внутренние переменные и регистры. В IL можно использовать внутренние переменные и регистры для временного хранения данных или промежуточных результатов операций. Они могут быть использованы для обработки данных внутри программы без воздействия на внешние устройства.

9. Циклы и переходы. IL позволяет создавать циклы и обеспечивает возможность выполнения повторяющихся операций. Также поддерживаются инструкции для управления переходами между различными состояниями программы.

10. Программные вызовы. IL может включать программные вызовы, позволяющие повторно использовать фрагменты кода и создавать более модульные программы. Это делает программирование более структурированным и обеспечивает возможность создания более сложных логических структур.

11. Директивы компилятора и системные функции. IL может содержать директивы компилятора и вызовы системных функций, которые обеспечивают дополнительные возможности программирования и взаимодействия с операционной системой контроллера.

Эти дополнительные возможности расширяют функциональность IL и делают его мощным инструментом для программирования логических контроллеров, позволяя разработчикам создавать сложные и эффективные программные решения для автоматизации промышленных процессов.

2.5. Sequential Function Chart (SFC): принципы и применение

Sequential Function Chart (SFC) представляет собой графический язык программирования, который используется для описания последовательности действий в промышленных процессах. В SFC программа представляется в виде графа, состоящего из различных шагов (стадий) и переходов между ними.

Основные принципы SFC включают в себя следующие.

1. Структура программы. Программа на SFC состоит из различных стадий, которые представляют собой отдельные части программы, выполняющие определенные действия или задачи. Стадии соединены переходами, которые определяют последовательность их выполнения.

2. Параллельность и последовательность. SFC позволяет описывать параллельное и последовательное выполнение действий в промышленном процессе. Это позволяет создавать сложные логические структуры, которые учитывают одновременное выполнение различных операций.

3. Условные переходы. В SFC можно определять условные переходы между стадиями, которые зависят от определенных условий или событий. Это позволяет создавать гибкие и адаптивные программы, которые могут реагировать на изменения внешних условий.

4. Переиспользование кода. SFC поддерживает возможность переиспользования кода путем создания библиотек стадий и переходов. Это упрощает разработку программы и повышает ее модульность и повторное использование.

SFC находит применение в различных областях промышленной автоматизации, таких как управление производственными процессами, технологическими установками и системами, транспортными и складскими системами и др.

SFC позволяет создавать сложные и надежные программы управления, которые эффективно управляют промышленными процессами и обеспечивают их безопасность и эффективность.

Применение Sequential Function Chart (SFC) можно рассмотреть на примере автоматизации процесса управления конвейерной системой в производстве.

1. Инициализация (Initialization). Первая стадия программы может быть посвящена инициализации системы. В этой стадии происходит запуск конвейерной ленты, активация датчиков и установка начальных параметров.

2. Загрузка материалов (Material Loading). После инициализации системы начинается стадия загрузки материалов на конвейерную ленту. В этой стадии программа ожидает сигнала о начале загрузки и активирует соответствующие механизмы для передачи материалов на ленту.

3. Транспортировка (Transportation). В этой стадии материалы перемещаются по конвейеру от начальной точки к конечной. Программа обеспечивает непрерывное движение ленты и контролирует скорость и направление движения материалов.

4. Выгрузка материалов (Material Unloading). По достижении конечной точки конвейерной системы начинается стадия выгрузки материалов. Программа активирует механизмы для выгрузки материалов с ленты и передачи их на следующий этап производственного процесса.

5. Остановка (Stop). После завершения цикла работы системы происходит остановка конвейера и подготовка к следующему циклу работы. Программа переводит систему в состояние ожидания новой команды или сигнала о начале работы.

Программа на Sequential Function Chart (SFC) позволяет удобно описывать последовательность действий в процессе управления конвейерной системой. Графическое представление программы делает ее легко визуализируемой и понятной для операторов и разработчиков, что упрощает ее создание, отладку и сопровождение.

Рассмотрим простой пример кода на Sequential Function Chart (SFC), который моделирует процесс управления освещением в помещении. В этом примере будет три стадии: «Включение», «Работа» и «Выключение».

```sfc
PROGRAM LightingControl
VAR
    SwitchOn: BOOL := FALSE;
    LightOn: BOOL := FALSE;
END_VAR

( Начальная стадия )
INITIAL_STEP Start
    ( Проверяем, включен ли выключатель )
    TRANSITION TO SwitchOn = TRUE BY SwitchOn;
END_STEP

( Стадия включения )
STEP TurnOn
    ( Включаем свет )
```

```
    LightOn := TRUE;
    ( Переходим к следующей стадии )
    TRANSITION TO Working;
END_STEP

( Рабочая стадия )
STEP Working
    ( Осуществляем рабочие действия )
    ( Здесь могут быть дополнительные операции )
    ( Переходим к следующей стадии )
    TRANSITION TO SwitchOn = FALSE BY NOT SwitchOn;
END_STEP

( Стадия выключения )
STEP TurnOff
    ( Выключаем свет )
    LightOn := FALSE;
    ( Переходим к начальной стадии )
    TRANSITION TO Start;
END_STEP
```

В этом коде:

– SwitchOn — это переменная, отображающая состояние выключателя (включен или выключен);

– LightOn — это переменная, отображающая состояние освещения (включено или выключено);

– программа начинается с начальной стадии Start, которая проверяет состояние выключателя;

– если выключатель включен, программа переходит в стадию TurnOn, включающую свет;

– затем программа переходит в рабочую стадию Working, где могут выполняться какие-либо дополнительные операции;

– если выключатель выключен во время работы, программа переходит в стадию TurnOff, выключающую свет;

– программа продолжает циклически выполнять эти стадии, пока не завершится или не будет принято другое условие перехода.

Этот пример демонстрирует основные концепции SFC и их применение для моделирования последовательности действий в промышленных процессах.

Рассмотрим другой пример кода на Sequential Function Chart (SFC), который моделирует процесс работы автоматической системы полива растений. В этом примере будем использовать несколько ста-

дий для управления поливом в зависимости от уровня влажности почвы.

```sfc
PROGRAM IrrigationSystem
VAR
    HumidityLevel: REAL := 0.0; ( Уровень влажности почвы
)
    PumpRunning: BOOL := FALSE; ( Состояние насоса:
запущен или остановлен )
END_VAR

( Начальная стадия )
INITIAL_STEP Start
    ( Проверяем уровень влажности )
    TRANSITION TO HumidityLevel >= 0.5 BY HumidityLevel;
END_STEP

( Стадия включения насоса )
STEP TurnOnPump
    ( Включаем насос )
    PumpRunning := TRUE;
    ( Переходим к стадии работы насоса )
    TRANSITION TO PumpRunning = FALSE;
END_STEP

( Стадия работы насоса )
STEP PumpWorking
    ( Ожидаем, пока уровень влажности не снизится )
    TRANSITION TO HumidityLevel < 0.5 BY HumidityLevel;
    ( Переходим к стадии выключения насоса )
    TRANSITION TO PumpRunning = TRUE;
END_STEP

( Стадия выключения насоса )
STEP TurnOffPump
    ( Выключаем насос )
    PumpRunning := FALSE;
    ( Переходим к начальной стадии )
    TRANSITION TO Start;
END_STEP
```

В этом коде:

– HumidityLevel — это переменная, отображающая уровень влажности почвы. В данном примере мы считаем, что нормальный уровень влажности составляет 0.5. Если уровень влажности превышает

этот порог, то земля считается достаточно влажной и полив не требуется;

– PumpRunning — это переменная, отображающая состояние насоса (запущен или остановлен);

– программа начинается с начальной стадии Start, которая проверяет уровень влажности почвы. Если уровень влажности выше или равен 0.5, программа переходит на стадию TurnOnPump, включающую насос;

– после включения насоса программа переходит на стадию PumpWorking, где насос продолжает работать, пока уровень влажности не снизится ниже 0.5. Затем программа переходит на стадию TurnOffPump, выключающую насос;

– программа циклически выполняет эти стадии в зависимости от уровня влажности почвы, обеспечивая оптимальный режим полива растений.

Рассмотрим сложный пример использования Sequential Function Chart (SFC) для управления процессом производства на предприятии. В этом примере будем моделировать автоматизированную линию сборки электронных устройств, которая включает несколько этапов: подготовка компонентов, сборка, тестирование и упаковка.

```sfc
PROGRAM AssemblyLine
VAR
    ComponentReady: BOOL := FALSE;  ( Готовность
компонентов )
    AssemblyInProgress: BOOL := FALSE; ( Процесс сборки в
процессе )
    TestingInProgress: BOOL := FALSE; ( Процесс
тестирования в процессе )
    Packaged: BOOL := FALSE; ( Упаковка завершена )
END_VAR

( Начальная стадия )
INITIAL_STEP Start
    ( Проверяем готовность компонентов )
    TRANSITION TO ComponentReady = TRUE BY Componen-
tReady;
END_STEP

( Подготовка компонентов )
STEP PrepareComponents
    ( Проводим подготовку компонентов )
    ( Здесь могут быть дополнительные операции )
    ( Переходим к следующей стадии )
```

```
    TRANSITION TO AssemblyInProgress = TRUE;
END_STEP

( Сборка устройства )
STEP Assembly
    ( Запускаем процесс сборки )
    ( Здесь могут быть дополнительные операции )
    ( Переходим к следующей стадии )
    TRANSITION TO TestingInProgress = TRUE;
END_STEP

( Тестирование устройства )
STEP Testing
    ( Запускаем процесс тестирования )
    ( Здесь могут быть дополнительные операции )
    ( Переходим к следующей стадии )
    TRANSITION TO Packaged = TRUE;
END_STEP

( Упаковка устройства )
STEP Packaging
    ( Запускаем процесс упаковки )
    ( Здесь могут быть дополнительные операции )
    ( Переходим к завершающей стадии )
    TRANSITION TO Start;
END_STEP
```

В этом коде:

– ComponentReady — это переменная, отображающая готовность компонентов для сборки;

– AssemblyInProgress — это переменная, отображающая состояние процесса сборки;

– TestingInProgress — это переменная, отображающая состояние процесса тестирования;

– Packaged — это переменная, отображающая завершение упаковки устройства.

Программа начинается с начальной стадии Start, которая проверяет готовность компонентов. Затем программа последовательно выполняет каждую стадию производственного процесса, переходя к следующей стадии после завершения предыдущей. Каждая стадия может содержать дополнительные операции и условия перехода в зависимости от конкретных требований производства.

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММ ДЛЯ PLC

3.1. Методологии структурного и модульного программирования

Разработка программ для программируемых логических контроллеров (PLC) требует применения эффективных методологий, таких как структурное и модульное программирование. Эти подходы помогают обеспечить надежность, гибкость и легкость сопровождения программного обеспечения.

Структурное программирование основано на использовании четко определенных блоков кода, каждый из которых выполняет одну задачу. Этот метод позволяет разбить сложные задачи на более мелкие и управляемые части. Программы, созданные с использованием структурного программирования, легче читать и понимать, что упрощает их отладку и сопровождение. В структурном программировании широко используются такие элементы, как условные операторы (IF-THEN-ELSE), циклы (FOR, WHILE) и процедуры.

Модульное программирование предполагает разделение программы на независимые модули, каждый из которых выполняет определенную функцию. Это позволяет разрабатывать и тестировать модули отдельно друг от друга, а затем интегрировать их в общую систему. Модульное программирование повышает повторное использование кода, так как один и тот же модуль может быть использован в различных проектах. Модули могут быть библиотеками функций, которые выполняют стандартные задачи, такие как управление вводом-выводом, обработка сигналов или выполнение математических вычислений [60–67].

Применение этих методологий при разработке программ для PLC обеспечивает более высокое качество программного обеспечения и упрощает процесс его создания. Структурное и модульное программирование позволяют создавать масштабируемые и легко адаптируемые программы, которые могут эволюционировать вместе с изменяющимися требованиями производства. Это особенно важно для промышленной автоматизации, где надежность и гибкость являются ключевыми факторами успеха.

Рассмотрим конкретный кейс разработки программы для PLC, управляющей системой контроля и управления температурой в промышленной печи.

В этом примере будем использовать структурное и модульное программирование для создания гибкой и легко сопровождаемой программы.

Описание задачи.

Промышленная печь должна поддерживать заданную температуру в течение определенного времени. Система управления должна:

1) считывать текущую температуру из датчика;

2) включать или выключать нагреватель в зависимости от текущей температуры;

3) поддерживать температуру в заданном диапазоне;

4) логировать данные о температуре и состоянии нагревателя.

Структурное программирование.

Разобьем задачу на несколько функциональных блоков:

1) чтение данных с датчика температуры;

2) управление нагревателем;

3) логирование данных;

4) основной цикл программы.

Модульное программирование.

Создадим модули для каждого функционального блока, чтобы их можно было разрабатывать, тестировать и модифицировать независимо.

Модуль: чтение данных с датчика температуры.

```structuredtext
FUNCTION_BLOCK ReadTemperature
VAR_OUTPUT
    CurrentTemperature: REAL;
END_VAR

( Имитация чтения температуры с датчика )
CurrentTemperature :=  ReadTemperatureSensor();
END_FUNCTION_BLOCK
```

Модуль: управление нагревателем.

```structuredtext
FUNCTION_BLOCK ControlHeater
VAR_INPUT
    CurrentTemperature: REAL;
    SetPointTemperature: REAL;
    Hysteresis: REAL;
END_VAR
VAR_OUTPUT
    HeaterState: BOOL;
END_VAR

IF CurrentTemperature < (SetPointTemperature — Hystere-
sis) THEN
```

```
    HeaterState := TRUE; ( Включить нагреватель )
ELSIF CurrentTemperature > (SetPointTemperature + Hyste-
resis) THEN
    HeaterState := FALSE; ( Выключить нагреватель )
END_IF
END_FUNCTION_BLOCK
```

Модуль: логирование данных.

```structuredtext
FUNCTION_BLOCK LogData
VAR_INPUT
    CurrentTemperature: REAL;
    HeaterState: BOOL;
END_VAR

( Запись данных в лог )
WriteLog(CurrentTemperature, HeaterState);
END_FUNCTION_BLOCK
```

Основной цикл программы:

```structuredtext
PROGRAM Main
VAR
    TemperatureSensor: ReadTemperature;
    HeaterController: ControlHeater;
    DataLogger: LogData;

    CurrentTemp: REAL;
    HeaterOn: BOOL;
    SetTemp: REAL := 200.0; ( Установленная температура )
    Hysteresis: REAL := 2.0; ( Гистерезис для управления )
END_VAR

( Основной цикл программы )
TemperatureSensor(); ( Чтение текущей температуры )
CurrentTemp := TemperatureSensor.CurrentTemperature;

HeaterController(CurrentTemp, SetTemp, Hysteresis);
( Управление нагревателем )
HeaterOn := HeaterController.HeaterState;

DataLogger(CurrentTemp, HeaterOn); ( Логирование данных )
END_PROGRAM
```

Пояснение.

1. Модуль ReadTemperature. Этот функциональный блок отвечает за чтение данных с датчика температуры. Функция ReadTemperatureSensor() симулирует процесс получения температуры.

2. Модуль ControlHeater. Этот блок управляет состоянием нагревателя на основе текущей температуры, установленной температуры и гистерезиса. Если текущая температура ниже установленной температуры минус гистерезис, нагреватель включается, если выше установленной температуры плюс гистерезис, нагреватель выключается.

3. Модуль LogData. Этот блок отвечает за логирование данных о текущей температуре и состоянии нагревателя. Функция WriteLog записывает данные в лог.

4. Основной цикл программы. Здесь происходит последовательное выполнение всех модулей. Сначала считывается текущая температура, затем на основе этих данных управляется нагреватель, и наконец, все данные логируются.

Этот пример демонстрирует, как структурное и модульное программирование можно использовать для создания гибкой, масштабируемой и легко сопровождаемой программы для управления промышленной печью.

В процессе разработки программ важно учитывать не только функциональные требования, но и аспекты безопасности, производительности и энергоэффективности [68–75]. Структурный подход помогает систематически учитывать эти аспекты на каждом этапе разработки. Модульный подход позволяет легко заменять или обновлять отдельные компоненты программы без необходимости изменения всей системы, что значительно сокращает время и затраты на обслуживание и модернизацию.

Рассмотрим более сложный кейс, включающий несколько зон нагрева в промышленной печи, которые необходимо контролировать и поддерживать на различных уровнях температуры. Дополнительно внедрим механизмы аварийного отключения при перегреве и систему уведомлений об отклонениях.

Описание задачи.

Промышленная печь имеет три зоны нагрева, каждая из которых должна поддерживать свою заданную температуру. Система управления должна считывать текущую температуру в каждой зоне, управлять соответствующими нагревателями, поддерживать температуру в заданном диапазоне, логировать данные и выдавать уведомления при критических отклонениях. Кроме того, при обнаружении перегрева система должна аварийно отключить соответствующую зону.

Структурное программирование.

Разобьем задачу на следующие функциональные блоки: чтение данных с датчиков температуры для каждой зоны, управление нагревателями каждой зоны, логирование данных, аварийное отключение при перегреве и система уведомлений.

Модульное программирование.

Создадим модули для каждого функционального блока, чтобы их можно было разрабатывать, тестировать и модифицировать независимо.

Пример кода.

Модуль: чтение данных с датчиков температуры.

```structuredtext
FUNCTION_BLOCK ReadTemperature
VAR_OUTPUT
    Zone1Temp: REAL;
    Zone2Temp: REAL;
    Zone3Temp: REAL;
END_VAR

Zone1Temp := ReadTemperatureSensor(1);
Zone2Temp := ReadTemperatureSensor(2);
Zone3Temp := ReadTemperatureSensor(3);
END_FUNCTION_BLOCK
```

Модуль: управление нагревателями.

```structuredtext
FUNCTION_BLOCK ControlHeater
VAR_INPUT
    CurrentTemperature: REAL;
    SetPointTemperature: REAL;
    Hysteresis: REAL;
END_VAR
VAR_OUTPUT
    HeaterState: BOOL;
END_VAR

IF CurrentTemperature < (SetPointTemperature — Hystere-
sis) THEN
    HeaterState := TRUE;
ELSIF CurrentTemperature > (SetPointTemperature + Hyste-
resis) THEN
    HeaterState := FALSE;
END_IF
END_FUNCTION_BLOCK
```

Модуль: логирование данных.

```structuredtext
FUNCTION_BLOCK LogData
VAR_INPUT
    Zone1Temp: REAL;
    Zone2Temp: REAL;
    Zone3Temp: REAL;
    Zone1HeaterState: BOOL;
    Zone2HeaterState: BOOL;
    Zone3HeaterState: BOOL;
END_VAR

WriteLog(1, Zone1Temp, Zone1HeaterState);
WriteLog(2, Zone2Temp, Zone2HeaterState);
WriteLog(3, Zone3Temp, Zone3HeaterState);
END_FUNCTION_BLOCK
```

Модуль: аварийное отключение при перегреве.

```structuredtext
FUNCTION_BLOCK EmergencyShutdown
VAR_INPUT
    CurrentTemperature: REAL;
    MaxTemperature: REAL;
END_VAR
VAR_OUTPUT
    Shutdown: BOOL;
END_VAR

IF CurrentTemperature > MaxTemperature THEN
    Shutdown := TRUE;
ELSE
    Shutdown := FALSE;
END_IF
END_FUNCTION_BLOCK
```

Модуль: система уведомлений.

```structuredtext
FUNCTION_BLOCK NotificationSystem
VAR_INPUT
    Zone1Temp: REAL;
    Zone2Temp: REAL;
    Zone3Temp: REAL;
    MaxTemperature: REAL;
END_VAR
```

```
IF Zone1Temp > MaxTemperature THEN
    SendNotification("Zone 1 Overheating", Zone1Temp);
END_IF

IF Zone2Temp > MaxTemperature THEN
    SendNotification("Zone 2 Overheating", Zone2Temp);
END_IF

IF Zone3Temp > MaxTemperature THEN
    SendNotification("Zone 3 Overheating", Zone3Temp);
END_IF
END_FUNCTION_BLOCK
```

Основной цикл программы:

```structuredtext
PROGRAM Main
VAR
    TemperatureSensor: ReadTemperature;
    HeaterController1: ControlHeater;
    HeaterController2: ControlHeater;
    HeaterController3: ControlHeater;
    DataLogger: LogData;
    EmergencyShutdown1: EmergencyShutdown;
    EmergencyShutdown2: EmergencyShutdown;
    EmergencyShutdown3: EmergencyShutdown;
    Notifier: NotificationSystem;

    Zone1Temp: REAL;
    Zone2Temp: REAL;
    Zone3Temp: REAL;
    Zone1HeaterOn: BOOL;
    Zone2HeaterOn: BOOL;
    Zone3HeaterOn: BOOL;
    SetTemp1: REAL := 200.0;
    SetTemp2: REAL := 220.0;
    SetTemp3: REAL := 240.0;
    Hysteresis: REAL := 2.0;
    MaxTemp: REAL := 250.0;
END_VAR

TemperatureSensor();
Zone1Temp := TemperatureSensor.Zone1Temp;
Zone2Temp := TemperatureSensor.Zone2Temp;
Zone3Temp := TemperatureSensor.Zone3Temp;
```

```
EmergencyShutdown1(Zone1Temp, MaxTemp);
EmergencyShutdown2(Zone2Temp, MaxTemp);
EmergencyShutdown3(Zone3Temp, MaxTemp);

IF NOT EmergencyShutdown1.Shutdown THEN
    HeaterController1(Zone1Temp, SetTemp1, Hysteresis);
    Zone1HeaterOn := HeaterController1.HeaterState;
ELSE
    Zone1HeaterOn := FALSE;
END_IF

IF NOT EmergencyShutdown2.Shutdown THEN
    HeaterController2(Zone2Temp, SetTemp2, Hysteresis);
    Zone2HeaterOn := HeaterController2.HeaterState;
ELSE
    Zone2HeaterOn := FALSE;
END_IF

IF NOT EmergencyShutdown3.Shutdown THEN
    HeaterController3(Zone3Temp, SetTemp3, Hysteresis);
    Zone3HeaterOn := HeaterController3.HeaterState;
ELSE
    Zone3HeaterOn := FALSE;
END_IF

DataLogger(Zone1Temp, Zone2Temp, Zone3Temp,
Zone1HeaterOn, Zone2HeaterOn, Zone3HeaterOn);
Notifier(Zone1Temp, Zone2Temp, Zone3Temp, MaxTemp);
END_PROGRAM
```

В этом усложненном примере каждая зона нагрева управляется отдельно, используя модульное программирование.

Модуль ReadTemperature считывает данные с датчиков для каждой зоны. Модули ControlHeater управляют нагревателями, поддерживая температуру в заданном диапазоне. Модуль LogData логирует данные для каждой зоны. Модули EmergencyShutdown обеспечивают аварийное отключение при перегреве, а модуль NotificationSystem отправляет уведомления при критических отклонениях температуры. В основном цикле программы все эти модули взаимодействуют, обеспечивая надежное и безопасное управление промышленной печью.

Таким образом, использование методологий структурного и модульного программирования при разработке программ для PLC является залогом создания надежных, гибких и легко сопровождаемых

систем автоматизации, способных эффективно выполнять свои функции в сложных и динамичных производственных средах [119–122].

Рассмотрим кейс управления конвейерной системой с несколькими секциями, каждая из которых оснащена своими датчиками и исполнительными механизмами. В этой системе важно обеспечить плавное движение товаров по конвейеру, своевременное обнаружение и устранение заторов, а также синхронизацию между секциями.

Описание задачи.

Конвейерная система состоит из трех секций. Каждая секция имеет датчики входа и выхода для отслеживания наличия товара, мотор для привода ленты и сигнальные лампы для индикации состояния. Система должна:

1) запускать моторы конвейера при наличии товара на входе;

2) останавливать моторы при отсутствии товара на выходе, чтобы избежать заторов;

3) синхронизировать работу моторов смежных секций для обеспечения плавного перехода товара;

4) обеспечивать аварийное отключение при обнаружении затора;

5) логировать данные о состоянии конвейера.

Структурное программирование.

Разобьем задачу на несколько функциональных блоков:

1) чтение данных с датчиков;

2) управление мотором конвейера;

3) обнаружение заторов;

4) логирование данных;

5) синхронизация между секциями.

Модульное программирование.

Создадим модули для каждого функционального блока, чтобы их можно было разрабатывать, тестировать и модифицировать независимо.

Пример кода.

Модуль: чтение данных с датчиков.

```
```structuredtext
FUNCTION_BLOCK ReadSensors
VAR_OUTPUT
    EntrySensor: BOOL;
    ExitSensor: BOOL;
END_VAR

EntrySensor := ReadSensor(SensorType := ENTRY);
ExitSensor := ReadSensor(SensorType := EXIT);
```
```

```
END_FUNCTION_BLOCK
```

Модуль: управление мотором конвейера.

```structuredtext
FUNCTION_BLOCK ControlMotor
VAR_INPUT
    EntrySensor: BOOL;
    ExitSensor: BOOL;
END_VAR
VAR_OUTPUT
    MotorState: BOOL;
END_VAR

IF EntrySensor AND NOT ExitSensor THEN
    MotorState := TRUE; ( Запуск мотора )
ELSE
    MotorState := FALSE; ( Остановка мотора )
END_IF
END_FUNCTION_BLOCK
```

Модуль: обнаружение заторов.

```structuredtext
FUNCTION_BLOCK DetectJam
VAR_INPUT
    EntrySensor: BOOL;
    ExitSensor: BOOL;
END_VAR
VAR_OUTPUT
    JamDetected: BOOL;
END_VAR

IF EntrySensor AND ExitSensor THEN
    JamDetected := TRUE; ( Обнаружен затор )
ELSE
    JamDetected := FALSE;
END_IF
END_FUNCTION_BLOCK
```

Модуль: логирование данных.

```structuredtext
FUNCTION_BLOCK LogData
VAR_INPUT
    SectionID: INT;
```

```
    EntrySensor: BOOL;
    ExitSensor: BOOL;
    MotorState: BOOL;
    JamDetected: BOOL;
END_VAR

WriteLog(SectionID, EntrySensor, ExitSensor, MotorState,
JamDetected);
END_FUNCTION_BLOCK
```

Модуль: синхронизация между секциями.

```structuredtext
FUNCTION_BLOCK SyncSections
VAR_INPUT
    CurrentSectionExitSensor: BOOL;
    NextSectionEntrySensor: BOOL;
END_VAR
VAR_OUTPUT
    SyncState: BOOL;
END_VAR

IF CurrentSectionExitSensor AND NOT NextSectionEntrySen-
sor THEN
    SyncState := TRUE; ( Синхронизация требуется )
ELSE
    SyncState := FALSE;
END_IF
END_FUNCTION_BLOCK
```

Основной цикл программы:

```structuredtext
PROGRAM Main
VAR
    SensorsSection1: ReadSensors;
    SensorsSection2: ReadSensors;
    SensorsSection3: ReadSensors;
    MotorController1: ControlMotor;
    MotorController2: ControlMotor;
    MotorController3: ControlMotor;
    JamDetector1: DetectJam;
    JamDetector2: DetectJam;
    JamDetector3: DetectJam;
    DataLogger1: LogData;
    DataLogger2: LogData;
    DataLogger3: LogData;
```

```
    SyncSection1To2: SyncSections;
    SyncSection2To3: SyncSections;

    Section1Entry: BOOL;
    Section1Exit: BOOL;
    Section2Entry: BOOL;
    Section2Exit: BOOL;
    Section3Entry: BOOL;
    Section3Exit: BOOL;
    MotorState1: BOOL;
    MotorState2: BOOL;
    MotorState3: BOOL;
    JamDetected1: BOOL;
    JamDetected2: BOOL;
    JamDetected3: BOOL;
    SyncState1To2: BOOL;
    SyncState2To3: BOOL;
END_VAR
( Чтение данных с датчиков )
SensorsSection1();
Section1Entry := SensorsSection1.EntrySensor;
Section1Exit := SensorsSection1.ExitSensor;

SensorsSection2();
Section2Entry := SensorsSection2.EntrySensor;
Section2Exit := SensorsSection2.ExitSensor;

SensorsSection3();
Section3Entry := SensorsSection3.EntrySensor;
Section3Exit := SensorsSection3.ExitSensor;

( Управление моторами )
MotorController1(Section1Entry, Section1Exit);
MotorState1 := MotorController1.MotorState;

MotorController2(Section2Entry, Section2Exit);
MotorState2 := MotorController2.MotorState;

MotorController3(Section3Entry, Section3Exit);
MotorState3 := MotorController3.MotorState;

( Обнаружение заторов )
JamDetector1(Section1Entry, Section1Exit);
JamDetected1 := JamDetector1.JamDetected;

JamDetector2(Section2Entry, Section2Exit);
JamDetected2 := JamDetector2.JamDetected;
```

```
JamDetector3(Section3Entry, Section3Exit);
JamDetected3 := JamDetector3.JamDetected;

( Логирование данных )
DataLogger1(1, Section1Entry, Section1Exit, MotorState1,
JamDetected1);
DataLogger2(2, Section2Entry, Section2Exit, MotorState2,
JamDetected2);
DataLogger3(3, Section3Entry, Section3Exit, MotorState3,
JamDetected3);

( Синхронизация между секциями )
SyncSection1To2(Section1Exit, Section2Entry);
SyncState1To2 := SyncSection1To2.SyncState;

SyncSection2To3(Section2Exit, Section3Entry);
SyncState2To3 := SyncSection2To3.SyncState;

( Управление синхронизацией )
IF SyncState1To2 THEN
    MotorState2 := TRUE;
END_IF

IF SyncState2To3 THEN
    MotorState3 := TRUE;
END_IF
END_PROGRAM
```

Схема взаимодействия модулей:

```dot
digraph G {
    node [shape=box];
    ReadSensors1 [label="ReadSensors Section 1"];
    ReadSensors2 [label="ReadSensors Section 2"];
    ReadSensors3 [label="ReadSensors Section 3"];
    ControlMotor1 [label="ControlMotor Section 1"];
    ControlMotor2 [label="ControlMotor Section 2"];
    ControlMotor3 [label="ControlMotor Section 3"];
    DetectJam1 [label="DetectJam Section 1"];
    DetectJam2 [label="DetectJam Section 2"];
    DetectJam3 [label="DetectJam Section 3"];
    LogData1 [label="LogData Section 1"];
    LogData2 [label="LogData Section 2"];
    LogData3 [label="LogData Section 3"];
```

```
    SyncSections1To2 [label="SyncSections 1 to 2"];
    SyncSections2To3 [label="SyncSections 2 to 3"];

    ReadSensors1 -> ControlMotor1;
    ReadSensors2 -> ControlMotor2;
    ReadSensors3 -> ControlMotor3;
    ControlMotor1 -> DetectJam1;
    ControlMotor2 -> DetectJam2;
    ControlMotor3 -> DetectJam3;
    DetectJam1 -> LogData1;
    DetectJam2 -> LogData2;
    DetectJam3 -> LogData3;
    ControlMotor1 -> SyncSections1To2;
    ControlMotor2 -> SyncSections1To2;
    ControlMotor2 -> SyncSections2To3;
    ControlMotor3 -> SyncSections2To3;
    SyncSections1To2 -> ControlMotor2;
    SyncSections2To3 -> ControlMotor3;
}
```

Объяснение схемы.

1. ReadSensors. Читает данные с датчиков входа и выхода для каждой секции конвейера.

2. ControlMotor. Управляет состоянием мотора каждой секции, основываясь на данных с датчиков.

3. DetectJam. Обнаруживает заторы, анализируя состояние датчиков входа и выхода.

4. LogData. Логирует данные о текущем состоянии конвейера, включая показания датчиков, состояние моторов и наличие заторов.

5. SyncSections. Синхронизирует работу моторов смежных секций, чтобы обеспечить плавный переход товаров между секциями.

Этот усложненный пример демонстрирует, как структурное и модульное программирование можно использовать для управления многосекционной конвейерной системой, обеспечивая надежность и гибкость управления.

В этом примере рассмотрена система управления многосекционным конвейером, состоящая из трех секций. Каждая секция оснащена датчиками на входе и выходе, мотором для привода ленты и сигнальными лампами для индикации состояния. Основной задачей системы является обеспечение плавного движения товаров по конвейеру, своевременное обнаружение и устранение заторов, а также синхронизация между секциями.

Система разбита на несколько функциональных блоков для структурного и модульного программирования. Первый функциональный блок ReadSensors отвечает за считывание данных с датчиков входа и выхода для каждой секции. Эти данные являются основой для дальнейших операций системы.

Блок ControlMotor управляет состоянием мотора каждой секции на основе данных с датчиков. Если на входе секции обнаружен товар, а на выходе нет, мотор запускается. В противном случае мотор останавливается, чтобы избежать заторов.

Блок DetectJam отвечает за обнаружение заторов. Он анализирует состояние датчиков входа и выхода и сигнализирует о заторе, если оба датчика одновременно обнаруживают товар. Это позволяет оперативно реагировать на проблемы и предотвращать накопление товаров на конвейере.

Блок LogData занимается логированием данных о текущем состоянии конвейера. Он записывает показания датчиков, состояние моторов и наличие заторов. Это важно для последующего анализа и оптимизации работы системы.

Блок SyncSections обеспечивает синхронизацию между смежными секциями конвейера. Он проверяет состояние датчиков выхода текущей секции и входа следующей секции и определяет необходимость синхронизации. Это позволяет обеспечить плавный переход товаров между секциями и избежать их накопления.

Основной цикл программы объединяет все функциональные блоки. Данные с датчиков считываются и передаются в блоки управления моторами, обнаружения заторов, логирования и синхронизации. Это позволяет системе работать как единое целое, обеспечивая надежное и эффективное управление конвейером.

В этом примере рассмотрена система управления многосекционным конвейером, состоящая из трех секций. Каждая секция оснащена датчиками на входе и выходе, мотором для привода ленты и сигнальными лампами для индикации состояния. Основной задачей системы является обеспечение плавного движения товаров по конвейеру, своевременное обнаружение и устранение заторов, а также синхронизация между секциями.

Система разбита на несколько функциональных блоков для структурного и модульного программирования. Первый функциональный блок ReadSensors отвечает за считывание данных с датчиков входа и выхода для каждой секции. Эти данные являются основой для дальнейших операций системы.

Блок ControlMotor управляет состоянием мотора каждой секции на основе данных с датчиков. Если на входе секции обнаружен товар, а на выходе нет, мотор запускается. В противном случае мотор останавливается, чтобы избежать заторов.

Блок DetectJam отвечает за обнаружение заторов. Он анализирует состояние датчиков входа и выхода и сигнализирует о заторе, если оба датчика одновременно обнаруживают товар. Это позволяет оперативно реагировать на проблемы и предотвращать накопление товаров на конвейере.

Блок LogData занимается логированием данных о текущем состоянии конвейера. Он записывает показания датчиков, состояние моторов и наличие заторов. Это важно для последующего анализа и оптимизации работы системы.

Блок SyncSections обеспечивает синхронизацию между смежными секциями конвейера. Он проверяет состояние датчиков выхода текущей секции и входа следующей секции и определяет необходимость синхронизации. Это позволяет обеспечить плавный переход товаров между секциями и избежать их накопления.

Основной цикл программы объединяет все функциональные блоки. Данные с датчиков считываются и передаются в блоки управления моторами, обнаружения заторов, логирования и синхронизации. Это позволяет системе работать как единое целое, обеспечивая надежное и эффективное управление конвейером.

Для улучшения системы управления многосекционным конвейером добавим блоки контроля температуры моторов, а также блок предиктивного обслуживания.

Этот блок будет считывать данные с датчиков температуры, установленных на моторах конвейера. Если температура мотора превышает допустимый предел, система должна немедленно остановить мотор и сигнализировать об аварийной ситуации.

Этот блок будет анализировать данные о работе конвейера, такие как время работы моторов, количество заторов и температурные режимы. На основе этих данных будет рассчитываться вероятность возникновения неисправностей и необходимость проведения технического обслуживания. Система будет предупреждать оператора о необходимости планового ремонта.

```
( Предиктивное обслуживание )
PredictiveMaint1(RunTime1, JamCount1, MotorTemperature1);
MaintRequired1 := PredictiveMaint1.MaintenanceRequired;
```

```
PredictiveMaint2(RunTime2, JamCount2, MotorTemperature2);
MaintRequired2 := PredictiveMaint2.MaintenanceRequired;

PredictiveMaint3(RunTime3, JamCount3, MotorTemperature3);
MaintRequired3 := PredictiveMaint3.MaintenanceRequired;

( Управление синхронизацией )
IF SyncState1To2 THEN
    MotorState2 := TRUE;
END_IF

IF SyncState2To3 THEN
    MotorState3 := TRUE;
END_IF

( Остановка моторов при перегреве )
IF Overheat1 THEN
    MotorState1 := FALSE;
END_IF

IF Overheat2 THEN
    MotorState2 := FALSE;
END_IF

IF Overheat3 THEN
    MotorState3 := FALSE;
END_IF
END_PROGRAM
```

Объяснение обновленной схемы.

1. ReadSensors. Читает данные с датчиков входа и выхода для каждой секции конвейера.

2. ControlMotor. Управляет состоянием мотора каждой секции, основываясь на данных с датчиков.

3. DetectJam. Обнаруживает заторы, анализируя состояние датчиков входа и выхода.

4. LogData. Логирует данные о текущем состоянии конвейера, включая показания датчиков, состояние моторов и наличие заторов.

5. SyncSections. Синхронизирует работу моторов смежных секций, чтобы обеспечить плавный переход товаров между секциями.

6. TemperatureControl. Контролирует температуру моторов и предотвращает перегрев, останавливая моторы в случае необходимости.

7. PredictiveMaintenance. Анализирует данные о работе конвейера для предиктивного обслуживания, предупреждая о необходимости проведения технического обслуживания.

Эти дополнения делают систему более надежной и предсказуемой, обеспечивая более высокий уровень автоматизации и управления процессом.

3.2. Отладка и тестирование программ для PLC

Процесс отладки и тестирования программ для PLC является критически важным этапом разработки, который позволяет убедиться в правильности работы системы и предотвратить возможные ошибки в режиме эксплуатации. Этот процесс включает несколько ключевых шагов и методов, которые обеспечивают качество и надежность программного обеспечения [76–79].

Этапы отладки и тестирования.

1. Синтаксическая проверка. На этом этапе проверяется правильность написания кода с точки зрения синтаксиса. Большинство сред программирования для PLC имеют встроенные инструменты для проверки синтаксических ошибок. Это позволяет быстро выявить и исправить ошибки написания кода до его загрузки на контроллер.

2. Моделирование и симуляция. Программы для PLC можно тестировать в симуляторе перед их загрузкой на физическое оборудование. Симуляторы позволяют создать виртуальную среду, в которой можно проверить работу логики программы без риска повреждения оборудования или создания опасных ситуаций. Это особенно полезно для сложных и критически важных систем.

3. Проверка функциональности. На этом этапе тестируется работа всех функциональных блоков и модулей программы. Проверяется, правильно ли реагирует система на входные сигналы, корректно ли выполняются логические операции и управляются ли выходные сигналы в соответствии с ожиданиями.

4. Тестирование на оборудовании. После успешного тестирования в симуляторе программа загружается на реальное оборудование. На этом этапе важно проверить работу системы в условиях, максимально приближенных к реальным условиям эксплуатации. Это позволяет выявить и устранить ошибки, которые могли остаться незамеченными в виртуальной среде.

5. Стресс-тестирование. Этот метод включает проверку работы системы в экстремальных условиях, например при максимальных нагрузках или в случае сбоя отдельных компонентов. Стресс-

тестирование позволяет убедиться, что система способна справляться с неожиданными ситуациями и продолжать корректно работать.

6. Регрессионное тестирование. При внесении изменений в программу важно убедиться, что новые изменения не нарушили работу уже протестированных и корректно работающих частей системы. Регрессия позволяет выявить и исправить такие ошибки до ввода системы в эксплуатацию.

Методы отладки.

1. Пошаговая отладка. Этот метод позволяет выполнять программу пошагово, проверяя состояние переменных и работу логических блоков на каждом этапе. Пошаговая отладка помогает детально изучить выполнение программы и найти ошибки в логике.

2. Логирование и мониторинг. Включение в программу функций логирования позволяет записывать состояние переменных, данные с датчиков и другие важные параметры во время выполнения программы. Эти данные можно анализировать для выявления причин ошибок и неполадок.

3. Использование отладочных инструментов. Современные среды программирования для PLC предоставляют широкий набор инструментов для отладки, включая графические средства визуализации, возможности мониторинга состояния системы в режиме реального времени и инструменты для анализа производительности.

Пример: отладка системы управления конвейером.

Рассмотрим пример отладки системы управления многосекционным конвейером, описанной ранее. После написания программы и проверки синтаксических ошибок следующим шагом будет моделирование работы конвейера в симуляторе. В симуляторе проверяется, правильно ли моторы реагируют на сигналы с датчиков, корректно ли обнаруживаются заторы и ведется ли логирование данных.

После успешного тестирования в симуляторе программа загружается на физический контроллер, установленный на тестовом конвейере. На этом этапе важно проверить, как система работает с реальными датчиками и моторами. Например, можно поместить товар на вход первой секции и наблюдать, как он перемещается по конвейеру. Проверяется, правильно ли моторы запускаются и останавливаются, синхронизируются ли секции и корректно ли система реагирует на заторы.

Если на этом этапе обнаруживаются ошибки, например моторы не запускаются или датчики неправильно считывают сигналы, необходимо провести пошаговую отладку. Можно выполнить программу пошагово, проверяя состояние переменных и сигналы с датчиков на

каждом шагу. Это поможет найти и исправить ошибки в логике программы.

Для повышения надежности можно провести стресс-тестирование, создавая условия с максимальной нагрузкой на конвейер, чтобы убедиться в стабильной работе системы. Также важно провести регрессионное тестирование после внесения любых изменений, чтобы убедиться, что они не нарушили работу уже протестированных частей программы.

Таким образом, отладка и тестирование программ для PLC являются многокомпонентным процессом, включающим проверку синтаксиса, моделирование, функциональное и стресс-тестирование, а также использование разнообразных методов отладки для обеспечения надежной и безопасной работы автоматизированных систем.

Процесс отладки и тестирования программ для PLC включает также проверку на соответствие техническим требованиям и стандартам безопасности. Это необходимо для гарантии, что система будет работать безопасно и надежно в условиях реального производства. Отладка программы начинается с проверки синтаксиса и логики на уровне кода. После этого проводится симуляция, где тестируются основные функции и проверяются возможные сценарии работы системы.

На этапе тестирования на оборудовании проверяются реальные входные и выходные сигналы, что позволяет убедиться в правильности работы системы в условиях реальной эксплуатации. Во время тестирования также проводится проверка взаимодействия всех компонентов системы, чтобы выявить и устранить возможные несовместимости. Стресс-тестирование позволяет проверить устойчивость системы к нагрузкам и выявить слабые места, которые могут привести к сбою.

Для повышения надежности и безопасности системы проводится регрессионное тестирование, которое помогает выявить ошибки, возникающие после внесения изменений в программу [112–118]. Этот этап позволяет убедиться, что новые функции и исправления не нарушили работу уже протестированных частей системы. Использование инструментов мониторинга и логирования помогает отслеживать состояние системы в режиме реального времени и быстро реагировать на возникающие проблемы.

Современные среды программирования для PLC предоставляют разнообразные инструменты для отладки, включая графические интерфейсы для визуализации логики программы, средства для пошаговой отладки и мониторинга состояния системы [80–86]. Эти инструменты значительно упрощают процесс отладки и помогают разработчикам быстро находить и исправлять ошибки (рис. 4). Также важно

проводить регулярные проверки и обновления программного обеспечения, чтобы поддерживать его актуальность и безопасность.

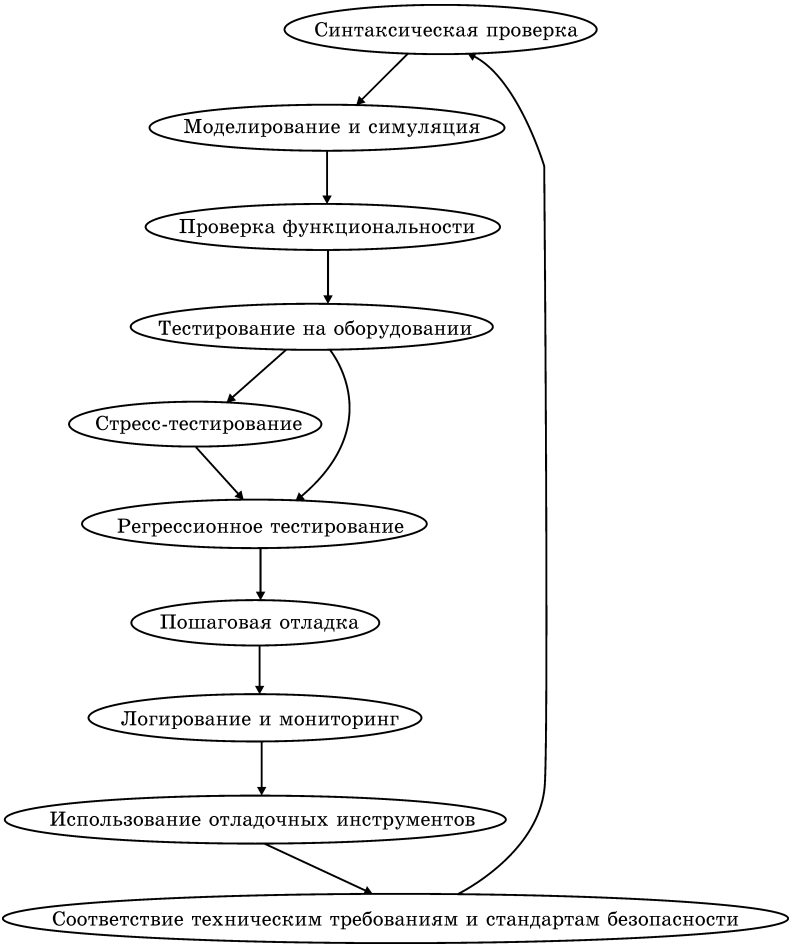

Рис. 4

Последовательность этапов отладки и тестирования программ для PLC

ГЛАВА 4. ПРОГРАММНЫЕ СРЕДЫ И ИНСТРУМЕНТЫ

4.1. Обзор программных сред: Siemens TIA Portal, Rockwell Automation Studio 5000, Schneider Electric EcoStruxure

Siemens TIA Portal представляет собой интегрированную среду разработки, предназначенную для программирования и настройки контроллеров Siemens. Это комплексное программное обеспечение объединяет различные инструменты, такие как SIMATIC S7, SIMATIC HMI и SIMATIC NET, что делает его мощным и удобным инструментом для создания систем автоматизации.

Siemens TIA (Totally Integrated Automation) Portal — это мощная интегрированная среда разработки, созданная для программирования и настройки промышленных контроллеров Siemens. Она объединяет в себе несколько инструментов, которые обеспечивают полный жизненный цикл разработки систем автоматизации.

Одним из ключевых компонентов TIA Portal является SIMATIC S7, который представляет собой семейство контроллеров Siemens для автоматизации различных производственных процессов. С помощью TIA Portal можно создавать и программировать различные типы контроллеров, такие как SIMATIC S7-1200, S7-1500 и др.

Кроме того, TIA Portal включает в себя инструменты для разработки пользовательских интерфейсов (SIMATIC HMI) и сетевых настроек (SIMATIC NET). Эти компоненты позволяют создавать графические интерфейсы оператора (HMI) для визуализации и управления процессами, а также настраивать сетевые соединения для обмена данными между контроллерами, устройствами ввода-вывода и другими компонентами системы.

Преимущество TIA Portal заключается в его интеграции и единой среде разработки, которая упрощает взаимодействие между различными компонентами системы автоматизации. Она обеспечивает единый интерфейс для программирования контроллеров, разработки интерфейсов оператора и настройки сетевых параметров, что повышает эффективность и удобство разработки. Кроме того, TIA Portal обеспечивает возможность симуляции и отладки разработанных программ непосредственно в среде разработки, что позволяет выявлять и устранять ошибки еще до внедрения системы.

Rockwell Automation Studio 5000 — это еще одна популярная программная среда, предназначенная для программирования контроллеров Allen-Bradley. Это интегрированная среда разработки, предоставляющая широкий набор инструментов, включая редакторы логических схем, функциональных блоков и текстовых программ.

Особенности и функциональные возможности Siemens TIA Portal.

1. Мощный редактор программного обеспечения. TIA Portal предоставляет интуитивно понятный и мощный графический интерфейс для программирования контроллеров, что упрощает процесс разработки (рис. 5).

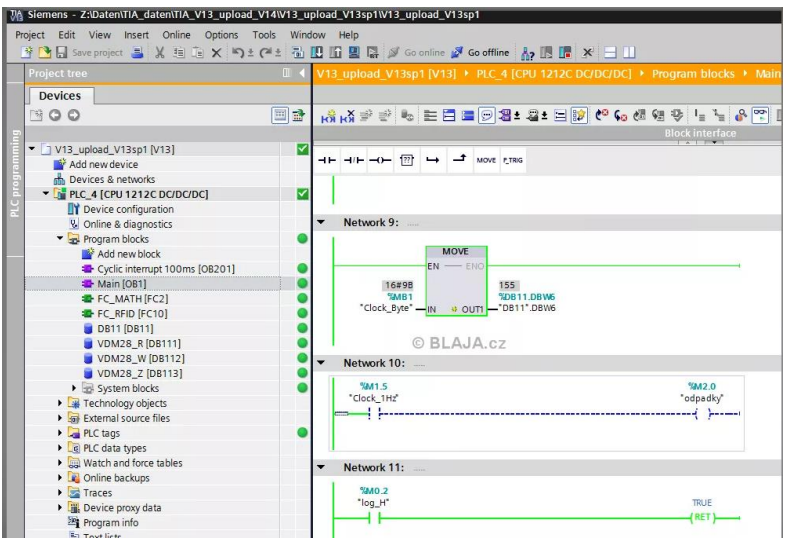

Рис. 5
TIA Portal

2. Библиотеки и шаблоны. Среда TIA Portal включает в себя библиотеки и шаблоны, которые содержат готовые функциональные блоки, элементы интерфейса и другие компоненты, что позволяет значительно ускорить процесс разработки за счет повторного использования кода.

3. Интеграция с другими системами. TIA Portal обеспечивает возможность интеграции с другими системами автоматизации и управления, такими как системы управления производственными процессами

(MES) и системы управления предприятием (ERP), что обеспечивает согласованность данных и процессов на разных уровнях предприятия.

4. Siemens TIA Portal соответствует множеству промышленных стандартов и протоколов связи, что делает его универсальным и готовым к интеграции с различным оборудованием и системами (рис. 6).

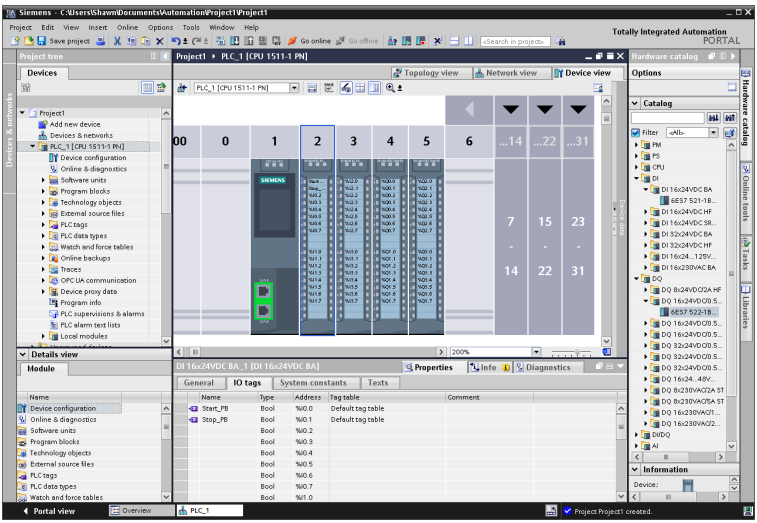

Рис. 6
Поддержка промышленных стандартов

5. Облачные сервисы. Некоторые версии TIA Portal поддерживают облачные сервисы, что позволяет разработчикам удаленно управлять и мониторить системы автоматизации, а также обеспечивает возможность совместной работы над проектами.

Такое дополнение поможет читателям получить более полное представление о возможностях и преимуществах программной среды Siemens TIA Portal.

Schneider Electric EcoStruxure — это платформа, предназначенная для разработки и управления системами автоматизации и управления зданиями (рис. 7).

Она включает в себя различные инструменты для программирования контроллеров Schneider Electric, таких как Modicon и других устройств (рис. 8).

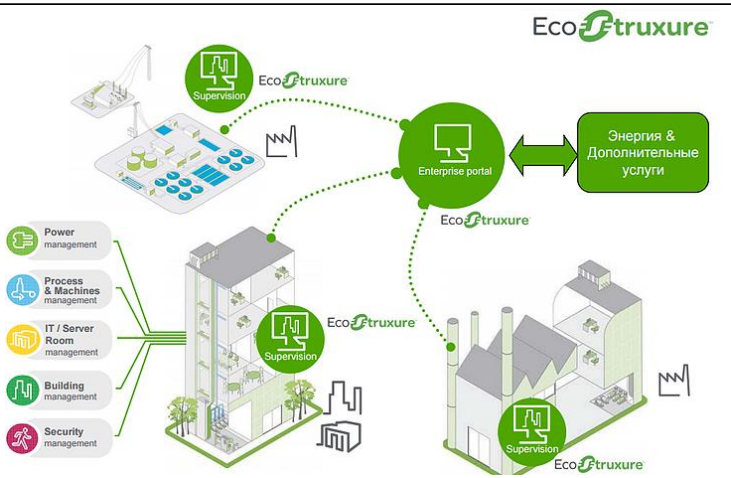

Рис. 7
Electric EcoStruxure (организационная структура)

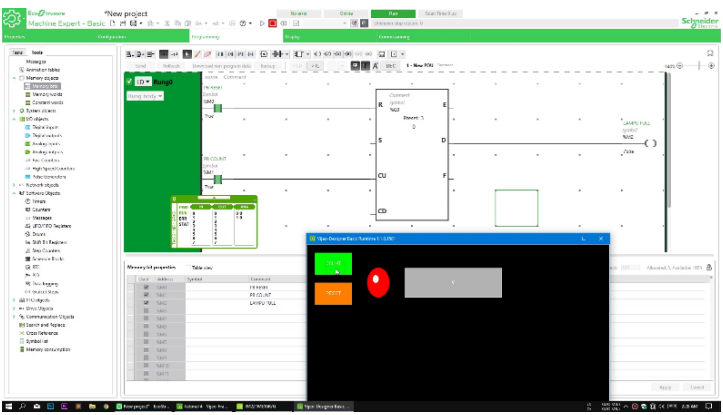

Рис. 8
Modicon IDE

Каждая из этих программных сред имеет свои особенности и преимущества, и выбор конкретной среды зависит от требований проекта, опыта разработчика и совместимости с используемым оборудованием [87–92].

Schneider Electric EcoStruxure — это интегрированная платформа, разработанная для создания и управления системами автоматизации и управления зданиями. Эта платформа объединяет в себе различные ин-

струменты и технологии, предназначенные для разработки программного обеспечения для промышленных контроллеров Schneider Electric, таких как Modicon, а также других устройств.

Основные компоненты Schneider Electric EcoStruxure.

1. EcoStruxure Control Expert. Это интегрированная среда разработки, предназначенная для программирования контроллеров Schneider Electric. Она обеспечивает широкий набор инструментов для разработки логических программ, конфигурации устройств ввода-вывода и настройки сетевых соединений.

2. Modicon M580. Это один из контроллеров Schneider Electric, который часто используется в системах автоматизации. Он предлагает высокую производительность, надежность и гибкость, а также поддерживает различные промышленные протоколы связи.

3. Другие устройства Schneider Electric. Кроме контроллеров Modicon, EcoStruxure поддерживает другие устройства Schneider Electric, такие как преобразователи частоты, панели оператора, датчики и пр. Все эти устройства могут интегрироваться и управляться с помощью единой платформы EcoStruxure.

Выбор конкретной программной среды Schneider Electric EcoStruxure зависит от требований проекта, опыта разработчика и совместимости с используемым оборудованием. Эта платформа предоставляет гибкость и масштабируемость для разработки различных систем автоматизации — от простых устройств до сложных интегрированных систем управления зданиями и производственных процессов.

4.2. Создание и симуляция проектов в TIA Portal

Создание и симуляция проектов в TIA Portal — это важный этап разработки систем автоматизации, который обеспечивает возможность проверки и отладки программного обеспечения до его реального внедрения.

Первым шагом является создание проекта в TIA Portal, где определяются основные параметры системы, включая типы контроллеров, используемые устройства ввода-вывода, сетевые настройки и пр. Затем проект разбивается на логические блоки, функциональные модули и другие компоненты программы.

После создания программы следует переход к симуляции, которая позволяет эмулировать работу системы на компьютере без привлечения реального оборудования. В процессе симуляции можно проверить правильность работы программы, отладить возможные ошибки и оптимизировать ее производительность.

Одним из инструментов симуляции в TIA Portal является встроенный симулятор PLC, который эмулирует работу программы на контроллере. Это позволяет разработчикам тестировать логику программы, отслеживать изменение значений переменных и реагировать на различные события.

Кроме того, TIA Portal поддерживает симуляцию визуализации (HMI), что позволяет проверить интерфейс оператора и визуализацию процессов без реального подключения панели оператора. Это обеспечивает полную проверку и отладку программы до ее внедрения на производственном оборудовании.

Создание и симуляция проектов в TIA Portal являются важным этапом разработки, который помогает обеспечить качество и надежность программного обеспечения перед его внедрением в реальную среду.

Ниже приведено пошаговое описание процесса создания и симуляции проектов в TIA Portal.

1. Открытие TIA Portal. Запустите программу TIA Portal на вашем компьютере.

2. Создание нового проекта. В меню выберите «Создать новый проект» или используйте сочетание клавиш (обычно <Ctrl + N>), чтобы начать новый проект.

3. Выбор типа устройства. Выберите тип контроллера или устройства, которое вы будете программировать в своем проекте. Например, это может быть контроллер SIMATIC S7 или другое устройство из линейки Siemens.

4. Настройка параметров проекта. Укажите параметры проекта, такие как название проекта, расположение файлов проекта и другие основные настройки.

5. Создание программы. После создания проекта перейдите к созданию программы. Это может включать в себя различные типы программирования, такие как логические схемы, структурированный текст или блоки функций, в зависимости от вашего предпочтения и требований проекта.

6. Добавление устройств и настройка ввода-вывода. Если ваш проект включает в себя использование устройств ввода-вывода, добавьте их в проект и настройте соответствующим образом.

7. Настройка сетевых соединений. Если в вашем проекте используются сетевые связи, настройте их параметры, чтобы обеспечить правильное взаимодействие между устройствами.

8. Симуляция проекта. После завершения программирования перейдите к симуляции проекта. В TIA Portal есть встроенный симулятор,

который позволяет эмулировать работу вашего проекта на компьютере без необходимости подключения реального оборудования.

9. Тестирование и отладка. При симуляции проекта тестируйте его на предмет правильности работы и отлаживайте любые обнаруженные ошибки или проблемы.

10. Оптимизация и доработка. После завершения симуляции и отладки проекта внесите любые необходимые корректировки или доработки, чтобы улучшить его производительность и эффективность.

11. Готовность к внедрению. После успешного завершения симуляции и отладки ваш проект готов к внедрению. Перенесите его на реальное оборудование и проверьте работоспособность в реальных условиях.

Это общий процесс создания и симуляции проектов в TIA Portal. Каждый из этих шагов может включать в себя дополнительные действия и настройки в зависимости от конкретных требований вашего проекта и используемого оборудования.

4.3. Управление технологическими процессами

Управление технологическими процессами в контексте программирования логических контроллеров включает в себя разработку и реализацию программного обеспечения, которое обеспечивает контроль, регулирование и мониторинг различных процессов в промышленности. В этом параграфе рассматриваются методы и инструменты, используемые для создания программ, которые управляют технологическими процессами.

Одним из ключевых аспектов управления технологическими процессами является разработка логики управления, которая определяет последовательность действий и условия, необходимые для эффективной работы системы. Для этого используются различные языки программирования, такие как схемы контактов, структурированный текст, функциональные блоки и др., которые позволяют инженерам создавать сложные алгоритмы управления.

Важным аспектом также является интеграция сенсоров, актуаторов и других устройств, которые собирают данные о состоянии процесса и позволяют контроллеру принимать соответствующие решения. Для этого используются различные коммуникационные протоколы и интерфейсы, такие как Profibus, Profinet, Ethernet/IP и др., которые обеспечивают связь между устройствами и контроллером.

Кроме того, управление технологическими процессами включает в себя разработку алгоритмов регулирования, которые обеспечи-

вают поддержание определенных параметров процесса в заданных пределах. Это может включать в себя автоматическое регулирование температуры, давления, уровня жидкости и других физических параметров с использованием различных методов управления, таких как ПИД-регуляторы, логические алгоритмы и др.

Наконец, важным аспектом является мониторинг и диагностика процессов, которые позволяют операторам отслеживать работу системы и быстро реагировать на любые неполадки или отклонения от заданных параметров. Для этого используются различные методы и инструменты, такие как визуализация данных, алармы, журналы событий и др., которые обеспечивают оперативное информирование о состоянии процесса и возможность оперативного вмешательства [93–97].

Визуализация и мониторинг процессов играют ключевую роль в обеспечении эффективного управления технологическими процессами. В этом параграфе рассматриваются методы визуализации данных и мониторинга процессов, а также инструменты, используемые для создания пользовательских интерфейсов и отображения информации о состоянии процесса.

Одним из основных методов визуализации данных является создание графических интерфейсов пользователя (GUI), которые предоставляют операторам возможность мониторинга и управления процессами с помощью интуитивно понятного интерфейса. Это может включать в себя отображение параметров процесса в виде графиков, диаграмм, числовых значений и других элементов, которые позволяют операторам быстро оценить текущее состояние системы.

Для создания пользовательских интерфейсов часто используются специализированные программные среды, такие как HMI/SCADA-системы (Human-Machine Interface / Supervisory Control and Data Acquisition), которые предоставляют широкий набор инструментов для создания графических интерфейсов, визуализации данных и управления процессами. Эти системы позволяют операторам создавать кастомизированные интерфейсы, а также настраивать алармы, журналы событий и другие функции, необходимые для мониторинга и управления процессами.

В дополнение для обеспечения эффективного мониторинга процессов часто используются системы аналитики данных и отчетности, которые позволяют анализировать и визуализировать большие объемы данных, собранных с различных устройств и датчиков. Это может включать в себя создание отчетов о производственной эффективности, анализ трендов и паттернов в данных, а также выявление аномалий и потенциальных проблем в процессах.

Интеграция визуализации и мониторинга процессов в программное обеспечение для PLC позволяет создавать полнофункциональные системы управления, которые обеспечивают оперативное реагирование на изменения в производственных процессах и повышают эффективность работы предприятия.

Визуализация и мониторинг процессов в контексте программирования логических контроллеров являются ключевыми аспектами для обеспечения эффективного управления и контроля за технологическими процессами в промышленности.

Это подразумевает создание пользовательских интерфейсов, которые позволяют операторам наблюдать и анализировать данные о состоянии процессов, а также принимать оперативные решения на основе этой информации.

Для визуализации данных и мониторинга процессов широко используются графические интерфейсы пользователя (GUI), которые отображают различные параметры процессов в виде графиков, диаграмм, числовых значений и других элементов. Операторы могут мониторить текущее состояние системы, а также анализировать исторические данные для выявления тенденций и аномалий.

Операторы могут настраивать интерфейсы в соответствии с требованиями производства, а также алармы и журналы событий для оперативного реагирования на неполадки.

Для обеспечения более глубокого анализа данных и мониторинга процессов используются системы аналитики данных и отчетности. Они позволяют анализировать большие объемы данных, выявлять тренды и паттерны, а также предсказывать возможные проблемы в процессах. Интеграция таких систем с программным обеспечением для PLC позволяет создавать комплексные системы управления, которые повышают эффективность производства и обеспечивают оперативное реагирование на изменения в процессах [98–103].

В промышленной автоматизации для визуализации и мониторинга процессов часто используются специализированные HMI/SCADA-системы. Ниже приведен пример простого кода на языке Structured Text (ST), который иллюстрирует основные шаги для взаимодействия с HMI/SCADA-системой:

```structured-text
PROGRAM Main
VAR
    StartButton: BOOL;          ( Переменная для кнопки
запуска процесса )
```

```
    StopButton: BOOL;            ( Переменная для кнопки
остановки процесса )
    Temperature: REAL;           ( Переменная для
измерения температуры )
    Pressure: REAL;              ( Переменная для
измерения давления )
END_VAR

( Основной цикл программы )
BEGIN
    ( Опрос состояния кнопок )
    StartButton := HMI_ReadButton("StartButton");
    StopButton := HMI_ReadButton("StopButton");

    ( Если нажата кнопка "Старт", выполнить процесс )
    IF StartButton THEN
        StartProcess();
    END_IF;

    ( Если нажата кнопка "Стоп", остановить процесс )
    IF StopButton THEN
        StopProcess();
    END_IF;

    ( Чтение данных с датчиков )
    Temperature := ReadTemperatureSensor();
    Pressure := ReadPressureSensor();

    ( Отправка данных на HMI/SCADA )
    HMI_SendData("Temperature", Temperature);
    HMI_SendData("Pressure", Pressure);
END_PROGRAM
```

В этом примере программа опрашивает состояние кнопок «Старт» и «Стоп» с HMI/SCADA-системы, считывает данные с датчиков температуры и давления, а затем отправляет их обратно на HMI/SCADA для отображения на графическом интерфейсе.

Давайте усложним пример, добавив взаимодействие с базой данных для хранения исторических данных о процессе. Мы также добавим функции для обработки аномалий данных и отправки уведомлений операторам через HMI/SCADA-систему. Ниже приведен обновленный пример на языке Structured Text.

```structured-text
PROGRAM Main
VAR
```

```
    StartButton: BOOL;          ( Переменная для кнопки
запуска процесса )
    StopButton: BOOL;           ( Переменная для кнопки
остановки процесса )
    Temperature: REAL;          ( Переменная для
измерения температуры )
    Pressure: REAL;             ( Переменная для
измерения давления )
    DatabaseConnection: BOOL;   ( Переменная для проверки
соединения с базой данных )
    AnomalyDetected: BOOL;      ( Переменная для
обнаружения аномалий в данных )
END_VAR

( Основной цикл программы )
BEGIN
    ( Опрос состояния кнопок )
    StartButton := HMI_ReadButton("StartButton");
    StopButton := HMI_ReadButton("StopButton");

    ( Если нажата кнопка "Старт", выполнить процесс )
    IF StartButton THEN
        StartProcess();
    END_IF;

    ( Если нажата кнопка "Стоп", остановить процесс )
    IF StopButton THEN
        StopProcess();
    END_IF;

    ( Чтение данных с датчиков )
    Temperature := ReadTemperatureSensor();
    Pressure := ReadPressureSensor();

    ( Отправка данных на HMI/SCADA )
    HMI_SendData("Temperature", Temperature);
    HMI_SendData("Pressure", Pressure);

    ( Подключение к базе данных )
    DatabaseConnection := ConnectToDatabase();

    ( Если соединение с базой данных установлено,
сохранить данные )
    IF DatabaseConnection THEN
        SaveDataToDatabase(Temperature, Pressure);
    END_IF;
```

```
    ( Обнаружение аномалий в данных )
    AnomalyDetected := DetectAnomalies(Temperature, Pres-
sure);

    ( Если обнаружены аномалии, отправить уведомление на
HMI/SCADA )
    IF AnomalyDetected THEN
        HMI_SendNotification("AnomalyDetected", TRUE);
    ELSE
        HMI_SendNotification("AnomalyDetected", FALSE);
    END_IF;
END_PROGRAM
```

Этот обновленный пример включает в себя функции для подключения к базе данных, сохранения данных в базе данных, обнаружения аномалий в данных и отправки уведомлений о возможных проблемах на HMI/SCADA-интерфейс.

Давайте рассмотрим другой кейс, связанный с управлением производственным процессом с использованием логических контроллеров. Допустим, у нас есть процесс смешивания нескольких ингредиентов для производства конечного продукта. В этом случае логический контроллер будет отвечать за управление всеми шагами этого процесса, начиная от загрузки ингредиентов до контроля качества конечного продукта.

Программа для логического контроллера может выглядеть следующим образом:

```structured-text
PROGRAM MixingProcess
VAR
    StartButton: BOOL;              ( Кнопка запуска
процесса )
    StopButton: BOOL;               ( Кнопка остановки
процесса )
    Ingredient1FlowRate: REAL;      ( Поток ингредиента 1 )
    Ingredient2FlowRate: REAL;      ( Поток ингредиента 2 )
    MixerSpeed: INT;                ( Скорость смешивания )
    ProcessRunning: BOOL;           ( Флаг, указывающий
на текущее состояние процесса )
END_VAR

( Основной цикл программы )
BEGIN
    ( Опрос состояния кнопок )
    StartButton := HMI_ReadButton("StartButton");
    StopButton := HMI_ReadButton("StopButton");
```

```
    ( Если нажата кнопка "Старт", запустить процесс )
    IF StartButton THEN
        ProcessRunning := TRUE;
        MixerSpeed := 50;    ( Начальная скорость
смешивания )
    END_IF;

    ( Если нажата кнопка "Стоп", остановить процесс )
    IF StopButton THEN
        ProcessRunning := FALSE;
    END_IF;

    ( Если процесс активен, управлять потоком
ингредиентов и скоростью смешивания )
    IF ProcessRunning THEN
        ( Чтение данных с датчиков )
        Ingredient1FlowRate := ReadFlowSen-
sor("Ingredient1");
        Ingredient2FlowRate := ReadFlowSen-
sor("Ingredient2");

        ( Управление смешиванием )
        IF Ingredient1FlowRate > 0 AND Ingredi-
ent2FlowRate > 0 THEN
            ( Если оба ингредиента поступают, увеличить
скорость смешивания )
            IF MixerSpeed < 100 THEN
                MixerSpeed := MixerSpeed + 10;
            END_IF;
        ELSE
            ( Если один из ингредиентов отсутствует,
уменьшить скорость смешивания )
            IF MixerSpeed > 0 THEN
                MixerSpeed := MixerSpeed — 10;
            END_IF;
        END_IF;

        ( Управление скоростью смешивания )
        SetMixerSpeed(MixerSpeed);
    ELSE
        ( Если процесс не активен, остановить
смешивание )
        SetMixerSpeed(0);
    END_IF;
END_PROGRAM
```

В этом примере программа управляет процессом смешивания, считывая данные с датчиков потока ингредиентов и управляя скоростью смешивания в зависимости от наличия ингредиентов. Также предусмотрены кнопки для запуска и остановки процесса.

Для оптимизации процесса смешивания можно добавить дополнительные функции.

1. Регулирование скорости смешивания. Вместо постоянного изменения скорости смешивания на фиксированный шаг, можно использовать обратную связь от датчиков качества продукта для автоматической регулировки скорости смешивания. Например, если датчик качества обнаруживает недостаточное смешивание, скорость смешивания автоматически увеличивается.

2. Предупреждения и аварии. Можно добавить систему предупреждений и аварийных ситуаций, которая будет автоматически реагировать на аномалии в процессе. Например, если датчик уровня ингредиентов обнаруживает их исчерпание, система может автоматически остановить процесс смешивания и отправить оператору сообщение о необходимости добавить новые ингредиенты.

3. Автоматическая калибровка датчиков. Для повышения точности измерений можно встроить функцию автоматической калибровки датчиков, которая будет периодически проверять и корректировать их показания.

4. Оптимизация расхода ингредиентов. Можно использовать алгоритмы оптимизации для минимизации расхода ингредиентов при достижении заданного качества конечного продукта.

Программу для управления процессом смешивания можно оптимизировать с использованием вышеописанных функций.

Оптимизируем программу для управления процессом смешивания, добавляя описанные функции. Ниже представлен обновленный код с пошаговыми комментариями.

```
```structured-text
PROGRAM MixingProcess
VAR
    StartButton: BOOL;                    ( Кнопка запуска
процесса )
    StopButton: BOOL;                     ( Кнопка
остановки процесса )
    Ingredient1FlowRate: REAL;            ( Поток
ингредиента 1 )
    Ingredient2FlowRate: REAL;            ( Поток
ингредиента 2 )
```
```

```
    MixerSpeed: INT;                    ( Скорость
смешивания )
    ProcessRunning: BOOL;               ( Флаг,
указывающий на текущее состояние процесса )
    QualitySensor: REAL;                ( Датчик качества
продукта )
    LevelSensor: BOOL;                  ( Датчик уровня
ингредиентов )
END_VAR

( Основной цикл программы )
BEGIN
    ( Опрос состояния кнопок )
    StartButton := HMI_ReadButton("StartButton");
    StopButton := HMI_ReadButton("StopButton");

    ( Если нажата кнопка "Старт", запустить процесс )
    IF StartButton THEN
        ProcessRunning := TRUE;
        MixerSpeed := 50;    ( Начальная скорость
смешивания )
    END_IF;

    ( Если нажата кнопка "Стоп", остановить процесс )
    IF StopButton THEN
        ProcessRunning := FALSE;
    END_IF;

    ( Если процесс активен, управлять потоком
ингредиентов и скоростью смешивания )
    IF ProcessRunning THEN
        ( Чтение данных с датчиков )
        Ingredient1FlowRate := ReadFlowSen-
sor("Ingredient1");
        Ingredient2FlowRate := ReadFlowSen-
sor("Ingredient2");
        QualitySensor := ReadQualitySensor();
        LevelSensor := ReadLevelSensor();

        ( Управление смешиванием )
        IF Ingredient1FlowRate > 0 AND Ingredi-
ent2FlowRate > 0 THEN
            ( Автоматическое регулирование скорости
смешивания в зависимости от качества продукта )
            IF QualitySensor < 0.95 THEN   ( Низкое
качество продукта )
                IF MixerSpeed < 100 THEN
```

```
                    MixerSpeed := MixerSpeed + 10;
                END_IF;
            ELSIF QualitySensor > 0.98 THEN    ( Высокое
качество продукта )
                IF MixerSpeed > 10 THEN
                    MixerSpeed := MixerSpeed — 10;
                END_IF;
            END_IF;

            ( Предупреждение об уровне ингредиентов )
            IF NOT LevelSensor THEN
                SendAlert("Low ingredient level detected.
Please refill.");
            END_IF;
        ELSE
            ( Если один из ингредиентов отсутствует,
остановить смешивание )
            SetMixerSpeed(0);
            ProcessRunning := FALSE;     ( Остановить
процесс )
        END_IF;

        ( Управление скоростью смешивания )
        SetMixerSpeed(MixerSpeed);
    ELSE
        ( Если процесс не активен, остановить смешивание
)
        SetMixerSpeed(0);
    END_IF;
END_PROGRAM
```

В этой версии программы добавлено автоматическое регулирование скорости смешивания в зависимости от качества продукта, а также предупреждение о низком уровне ингредиентов. Теперь программа более эффективно управляет процессом смешивания и предупреждает оператора о любых проблемах в процессе.

Для дальнейшего улучшения системы управления процессом смешивания можно внедрить нечеткие логические алгоритмы. Нечеткая логика позволит более гибко управлять скоростью смешивания и потоком ингредиентов в зависимости от нечетких условий, таких как качество продукта или уровень ингредиентов. Можно это сделать следующим образом.

1. Нечеткое управление скоростью смешивания. Мы можем определить нечеткие правила, основанные на нечетких переменных, таких как качество продукта, и использовать их для определения оп-

тимальной скорости смешивания. Например, если качество продукта считается «средним», то скорость смешивания будет установлена на среднем уровне.

2. Нечеткое управление потоком ингредиентов. Также мы можем применить нечеткую логику для определения оптимальных потоков ингредиентов в зависимости от текущих условий процесса. Например, если качество продукта требует большего количества одного из ингредиентов, нечеткая система может автоматически увеличить его потом.

3. Учет неопределенности. Нечеткая логика также позволяет учитывать неопределенность в системе, что особенно важно при работе с реальными данными и датчиками, подверженными шумам и погрешностям.

Давайте реализуем нечеткий контроллер скорости смешивания на языке Structured Text, используя библиотеку нечеткой логики для PLC. Для этого нам потребуется определить нечеткие переменные (например, качество продукта) и нечеткие правила, определяющие, как должна изменяться скорость смешивания в зависимости от значения этой переменной.

Ниже представлен пример кода для реализации нечеткого контроллера скорости смешивания:

```
```structured-text
PROGRAM FuzzyMixingController
VAR
    ProductQuality: REAL;               ( Качество
продукта )
    MixerSpeed: REAL;                   ( Скорость
смешивания )
    FuzzyLib: FUZZY_LIBRARY;            ( Библиотека
нечеткой логики )
END_VAR

( Определение нечетких переменных )
FuzzyLib.DEFINE_VARIABLE("ProductQuality", 0.0, 1.0,
0.01);   ( Качество продукта от 0 до 1 )

( Определение нечетких множеств и функций принадлежности
)
FuzzyLib.DEFINE_TRIANGLE_SET("LowQuality", 0.0, 0.2, 0.4);
FuzzyLib.DEFINE_TRIANGLE_SET("MediumQuality", 0.3, 0.5,
0.7);
FuzzyLib.DEFINE_TRIANGLE_SET("HighQuality", 0.6, 0.8, 1.0);
```
```

```
( Определение нечетких правил )
FuzzyLib.DEFINE_RULE("IncreaseSpeed", "IF ProductQuality
IS LowQuality THEN MixerSpeed IS Medium;");
FuzzyLib.DEFINE_RULE("DecreaseSpeed", "IF ProductQuality
IS HighQuality THEN MixerSpeed IS Low;");

( Основной цикл программы )
BEGIN
    ( Чтение значения качества продукта )
    ProductQuality := ReadProductQualitySensor();

    ( Определение скорости смешивания на основе нечетких
правил )
    FuzzyLib.EVALUATE("IncreaseSpeed");
    FuzzyLib.EVALUATE("DecreaseSpeed");

    ( Установка скорости смешивания )
    SetMixerSpeed(MixerSpeed);
END_PROGRAM
```

В этом примере мы определили переменную ProductQuality, которая представляет собой качество продукта от 0 до 1, затем нечеткие множества (например, LowQuality, MediumQuality, HighQuality) и соответствующие им функции принадлежности, далее нечеткие правила, которые определяют, как должна изменяться скорость смешивания в зависимости от качества продукта.

Этот пример демонстрирует принципы использования нечеткой логики для управления процессом смешивания на PLC.

Добавление нечетких алгоритмов позволит нам создать более адаптивную и гибкую систему управления, способную эффективно адаптироваться к различным условиям и изменениям в процессе смешивания.

Давайте усложним наш нечеткий контроллер скорости смешивания, добавив больше нечетких переменных и правил для более точного управления процессом. Мы можем добавить дополнительные параметры, такие как температура смеси, время смешивания и уровень вязкости продукта, и использовать их для определения оптимальной скорости смешивания.

Мы можем расширить наш код следующим образом:

```structured-text
PROGRAM AdvancedFuzzyMixingController
VAR
```

```
    ProductQuality: REAL;                ( Качество
продукта )
    MixerSpeed: REAL;                    ( Скорость
смешивания )
    Temperature: REAL;                   ( Температура
смеси )
    MixingTime: TIME;                   ( Время смешивания )
    Viscosity: REAL;                     ( Уровень вязкости
продукта )
    FuzzyLib: FUZZY_LIBRARY;             ( Библиотека
нечеткой логики )
END_VAR

( Определение нечетких переменных )
FuzzyLib.DEFINE_VARIABLE("ProductQuality", 0.0, 1.0,
0.01);   ( Качество продукта от 0 до 1 )
FuzzyLib.DEFINE_VARIABLE("Temperature", 0.0, 100.0, 1.0);
( Температура в градусах Цельсия )
FuzzyLib.DEFINE_VARIABLE("MixingTime", 0, 3600, 1);
( Время смешивания в секундах )
FuzzyLib.DEFINE_VARIABLE("Viscosity", 0.0, 10.0, 0.1);
( Уровень вязкости от 0 до 10 )

( Определение нечетких множеств и функций принадлежности
для каждой переменной )

( Определение нечетких правил, учитывающих все переменные
)

( Основной цикл программы )
BEGIN
    ( Чтение значений переменных )
    ProductQuality := ReadProductQualitySensor();
    Temperature := ReadTemperatureSensor();
    MixingTime := ReadMixingTimeSensor();
    Viscosity := ReadViscositySensor();

    ( Определение скорости смешивания на основе нечетких
правил )

    ( Установка скорости смешивания )
    SetMixerSpeed(MixerSpeed);
END_PROGRAM
```

В этом обновленном коде мы добавили дополнительные нечеткие переменные для температуры, времени смешивания и уровня вяз-

кости продукта. Затем мы определили соответствующие им нечеткие множества и функции принадлежности, а также нечеткие правила, которые учитывают все эти переменные для определения скорости смешивания. Это позволит нам создать более точный и адаптивный контроллер скорости смешивания, способный учитывать различные условия и параметры процесса.

Давайте усложним наш контроллер, добавив возможность адаптивного изменения параметров нечеткой логики в режиме реального времени на основе обратной связи от процесса смешивания. Это позволит нам создать более гибкий и точный контроллер, способный адаптироваться к изменяющимся условиям производства.

```structured-text
PROGRAM AdvancedAdaptiveFuzzyMixingController
VAR
    ProductQuality: REAL;               ( Качество
продукта )
    MixerSpeed: REAL;                   ( Скорость
смешивания )
    Temperature: REAL;                  ( Температура
смеси )
    MixingTime: TIME;                  ( Время смешивания )
    Viscosity: REAL;                    ( Уровень вязкости
продукта )
    FuzzyLib: FUZZY_LIBRARY;            ( Библиотека
нечеткой логики )
    Feedback: REAL;                     ( Обратная связь
от процесса смешивания )
END_VAR

( Определение нечетких переменных )
FuzzyLib.DEFINE_VARIABLE("ProductQuality", 0.0, 1.0,
0.01);   ( Качество продукта от 0 до 1 )
FuzzyLib.DEFINE_VARIABLE("Temperature", 0.0, 100.0, 1.0);
( Температура в градусах Цельсия )
FuzzyLib.DEFINE_VARIABLE("MixingTime", 0, 3600, 1);
( Время смешивания в секундах )
FuzzyLib.DEFINE_VARIABLE("Viscosity", 0.0, 10.0, 0.1);
( Уровень вязкости от 0 до 10 )

( Определение нечетких множеств и функций принадлежности
для каждой переменной )

( Определение нечетких правил )
```

```
( Основной цикл программы )
BEGIN
    REPEAT
        ( Чтение значений переменных )
        ProductQuality := ReadProductQualitySensor();
        Temperature := ReadTemperatureSensor();
        MixingTime := ReadMixingTimeSensor();
        Viscosity := ReadViscositySensor();

        ( Определение скорости смешивания на основе
нечетких правил )

        ( Установка скорости смешивания )
        SetMixerSpeed(MixerSpeed);

        ( Получение обратной связи от процесса смешивания
)
        Feedback := ReadFeedbackSensor();

        ( Адаптация параметров нечеткой логики на основе
обратной связи )
        AdaptFuzzyLogicParameters(FuzzyLib, Feedback);

        ( Пауза между итерациями )
        WAIT 1 SECOND;
    UNTIL FALSE;
END_PROGRAM
```

В этом улучшенном коде мы добавили переменную Feedback, которая представляет собой обратную связь от процесса смешивания. Затем мы используем эту обратную связь для адаптации параметров нечеткой логики в режиме реального времени, что позволяет нам улучшить производительность и точность контроллера.

Для внедрения механизмов блокирования в наш контроллер мы можем добавить соответствующие проверки и действия в основной цикл программы. В случае возникновения блокирующей ситуации, мы можем приостановить работу контроллера и отправить уведомление об этом оператору.

```structured-text
PROGRAM AdvancedAdaptiveFuzzyMixingController
VAR
    ProductQuality: REAL;              ( Качество
продукта )
    MixerSpeed: REAL;                  ( Скорость
смешивания )
```

```
    Temperature: REAL;                    ( Температура
смеси )
    MixingTime: TIME;                    ( Время смешивания )
    Viscosity: REAL;                      ( Уровень вязкости
продукта )
    FuzzyLib: FUZZY_LIBRARY;              ( Библиотека
нечеткой логики )
    Feedback: REAL;                       ( Обратная связь
от процесса смешивания )
    BlockFlag: BOOL := FALSE;            ( Флаг блокировки )
END_VAR

( Определение нечетких переменных и правил )

( Основной цикл программы )
BEGIN
    REPEAT
        IF NOT BlockFlag THEN
            ( Чтение значений переменных )
            ProductQuality := ReadProductQualitySensor();
            Temperature := ReadTemperatureSensor();
            MixingTime := ReadMixingTimeSensor();
            Viscosity := ReadViscositySensor();

            ( Определение скорости смешивания на основе
нечетких правил )

            ( Установка скорости смешивания )
            SetMixerSpeed(MixerSpeed);

            ( Получение обратной связи от процесса
смешивания )
            Feedback := ReadFeedbackSensor();

            ( Адаптация параметров нечеткой логики на
основе обратной связи )
            AdaptFuzzyLogicParameters(FuzzyLib, Feed-
back);
        ELSE
            ( Отправить уведомление об оперативной
блокировке )
            SendOperatorNotification("Блокировка!
Необходимо решить проблему.");
        END_IF

        ( Пауза между итерациями )
        WAIT 1 SECOND;
```

```
    UNTIL FALSE;
END_PROGRAM
```

В этой версии кода мы добавили переменную BlockFlag, которая указывает, заблокирован ли контроллер. Если контроллер заблокирован, он не выполняет основные операции, а вместо этого отправляет уведомление оператору. Теперь мы можем активировать или деактивировать блокировку в зависимости от обстоятельств, таких как аварийные ситуации или техническое обслуживание.

Для эффективного управления процессом смешивания материалов важно иметь механизмы контроля и регулировки параметров. В данной программе мы также реализуем механизмы адаптации параметров нечеткой логики на основе обратной связи от процесса смешивания. Это позволит нам динамически изменять стратегии управления в зависимости от текущих условий и требований к качеству продукции.

Для того чтобы оптимизировать процесс смешивания и повысить качество продукции, можно интегрировать нечеткие алгоритмы управления. Они позволят учитывать неопределенность и нелинейность процесса смешивания, что особенно важно в условиях изменяющихся параметров среды и сырья.

Дополнительно, можно добавить механизмы самодиагностики и аварийного управления. Это позволит системе автоматически реагировать на возможные отклонения и неполадки в работе, минимизируя риски производственных простоев и повреждения оборудования.

Рассмотрим ситуацию, когда качество продукции начинает снижаться из-за неправильной настройки параметров смешивания. Это может быть вызвано изменением характеристик сырья, неисправностью оборудования или ошибкой в программе управления. В таком случае контроллер должен обнаружить и диагностировать проблему, а затем принять меры для ее устранения.

Проблемный кейс может быть реализован следующим образом.

1. Считывание данных о качестве продукции и сравнение их с установленными стандартами.

2. Проверка работы датчиков и исполнительных механизмов на наличие неисправностей.

3. Анализ данных обратной связи от процесса смешивания и сравнение их с ожидаемыми значениями.

4. В случае обнаружения отклонений или несоответствий активация режима аварийной остановки и отправка уведомления оператору.

5. Запуск процедур автоматической диагностики и самодиагностики для выявления причины проблемы.

6. Адаптация параметров нечеткой логики управления в соответствии с новыми условиями работы и восстановление работы системы.

Решение проблемного кейса требует комплексного подхода, включающего как технические, так и программные меры. Важно, чтобы контроллер был способен быстро реагировать на изменения в окружающей среде и эффективно адаптироваться к новым условиям работы для обеспечения стабильного качества продукции.

Начнем формализацию проблемного кейса.

1. Считывание данных о качестве продукции:

– контроллер считывает данные о качестве продукции с помощью соответствующих датчиков или измерительных устройств;

– эти данные передаются контроллеру для анализа.

2. Проверка работы датчиков и исполнительных механизмов:

– контроллер проверяет работу датчиков, определяющих параметры сырья и качество продукции;

– он также проверяет исполнительные механизмы, ответственные за смешивание компонентов.

3. Анализ данных обратной связи:

– контроллер анализирует данные обратной связи от процесса смешивания, полученные от датчиков и измерительных устройств;

– сравнивает эти данные с установленными стандартами качества продукции.

4. Активация режима аварийной остановки:

– в случае обнаружения отклонений или несоответствий контроллер активирует режим аварийной остановки;

– это приводит к остановке процесса смешивания и предотвращает выпуск продукции низкого качества.

5. Запуск процедур диагностики и самодиагностики:

– контроллер запускает процедуры диагностики для выявления причины проблемы;

– он также запускает самодиагностику для проверки своего собственного состояния и работоспособности.

6. Адаптация параметров управления:

– после выявления причины проблемы контроллер адаптирует параметры управления в соответствии с новыми условиями;

– это может включать в себя коррекцию скорости смешивания, изменение пропорций компонентов или переключение на альтернативные режимы работы.

Этот процесс обеспечивает систему автоматизации возможностью обнаруживать и реагировать на возможные проблемы в процессе смешивания, что в конечном итоге способствует повышению качества продукции и эффективности производства.

Начнем формализацию проблемного кейса.

1. Считывание данных о качестве продукции:

– контроллер считывает данные о качестве продукции с помощью соответствующих датчиков или измерительных устройств;

– эти данные передаются контроллеру для анализа.

2. Проверка работы датчиков и исполнительных механизмов:

– контроллер проверяет работу датчиков, определяющих параметры сырья и качество продукции;

– он также проверяет исполнительные механизмы, ответственные за смешивание компонентов.

3. Анализ данных обратной связи:

– контроллер анализирует данные обратной связи от процесса смешивания, полученные от датчиков и измерительных устройств;

– сравнивает эти данные с установленными стандартами качества продукции.

4. Активация режима аварийной остановки:

– в случае обнаружения отклонений или несоответствий контроллер активирует режим аварийной остановки;

– это приводит к остановке процесса смешивания и предотвращает выпуск продукции низкого качества.

5. Запуск процедур диагностики и самодиагностики:

– контроллер запускает процедуры диагностики для выявления причины проблемы;

– он также запускает самодиагностику для проверки собственного состояния и работоспособности.

6. Адаптация параметров управления:

– после выявления причины проблемы контроллер адаптирует параметры управления в соответствии с новыми условиями;

– это может включать в себя коррекцию скорости смешивания, изменение пропорций компонентов или переключение на альтернативные режимы работы.

Этот процесс обеспечивает систему автоматизации возможностью обнаруживать и реагировать на возможные проблемы в процессе смешивания, что в конечном итоге способствует повышению качества продукции и эффективности производства.

Для реализации проблемного кейса в программе управления PLC на языке Structured Text, мы создадим модуль, который будет

обрабатывать данные с датчиков, проверять их на соответствие заданным стандартам, инициировать аварийную остановку при необходимости и адаптировать параметры процесса. Ниже представлен пример кода:

```structured-text
PROGRAM ProcessControl
VAR
    ProductQuality: REAL;          // Текущее качество
продукта
    QualityStandard: REAL := 0.95; // Стандарт качества
```

Конечно, давайте реализуем более подробный код для описанного проблемного кейса на языке Structured Text. Код будет включать считывание данных с датчиков, проверку на соответствие стандартам, аварийную остановку, диагностику и адаптацию параметров.

```structured-text
PROGRAM ProcessControl
VAR
    // Переменные для входных данных
    ProductQuality: REAL;          // Текущее качество
продукта
    Temperature: REAL;             // Текущая температура
смеси
    MixingTime: TIME;              // Время смешивания
    Viscosity: REAL;               // Текущая вязкость
продукта
    Feedback: REAL;                // Обратная связь от
процесса смешивания

    // Переменные для установленных стандартов и
параметров
    QualityStandard: REAL := 0.95; // Стандарт качества
продукта
    TemperatureLimit: REAL := 80.0; // Максимально
допустимая температура
    MaxMixingTime: TIME := T30m;  // Максимальное время
смешивания
    ViscosityLimit: REAL := 5.0;   // Максимально
допустимая вязкость

    // Переменные для управления
    MixerSpeed: REAL;              // Скорость смешивания
    ProcessRunning: BOOL;          // Флаг, указывающий
на текущее состояние процесса
    BlockFlag: BOOL := FALSE;      // Флаг блокировки
процесса
```

```
    // Переменные для диагностики
    SensorFault: BOOL;              // Флаг неисправности
датчика
    DiagnosisMessage: STRING[255]; // Сообщение
диагностики

    // Переменные для нечеткой логики
    FuzzyLib: FUZZY_LIBRARY;        // Библиотека нечеткой
логики

END_VAR

( Определение нечетких переменных и правил )
FuzzyLib.DEFINE_VARIABLE("ProductQuality", 0.0, 1.0,
0.01);
FuzzyLib.DEFINE_VARIABLE("Temperature", 0.0, 100.0, 1.0);
FuzzyLib.DEFINE_VARIABLE("MixingTime", 0, 3600, 1);
FuzzyLib.DEFINE_VARIABLE("Viscosity", 0.0, 10.0, 0.1);

FuzzyLib.DEFINE_TRIANGLE_SET("LowQuality", 0.0, 0.2,
0.4);
FuzzyLib.DEFINE_TRIANGLE_SET("MediumQuality", 0.4, 0.6,
0.8);
FuzzyLib.DEFINE_TRIANGLE_SET("HighQuality", 0.8, 1.0,
1.2);

FuzzyLib.DEFINE_RULE("IncreaseSpeed", "IF ProductQuality
IS LowQuality THEN MixerSpeed IS High;");
FuzzyLib.DEFINE_RULE("DecreaseSpeed", "IF ProductQuality
IS HighQuality THEN MixerSpeed IS Low;");

( Основной цикл программы )
BEGIN
    REPEAT
        IF NOT BlockFlag THEN
            // Чтение данных с датчиков
            ProductQuality := ReadProductQualitySensor();
            Temperature := ReadTemperatureSensor();
            MixingTime := ReadMixingTimeSensor();
            Viscosity := ReadViscositySensor();
            Feedback := ReadFeedbackSensor();

            // Проверка на соответствие стандартам
            IF ProductQuality < QualityStandard THEN
                BlockFlag := TRUE;
```

```
                DiagnosisMessage := "Качество продукта
ниже стандарта!";
            ELSIF Temperature > TemperatureLimit THEN
                BlockFlag := TRUE;
                DiagnosisMessage := "Температура смеси
выше допустимой!";
            ELSIF MixingTime > MaxMixingTime THEN
                BlockFlag := TRUE;
                DiagnosisMessage := "Время смешивания
превышено!";
            ELSIF Viscosity > ViscosityLimit THEN
                BlockFlag := TRUE;
                DiagnosisMessage := "Вязкость продукта
выше допустимой!";
            END_IF;

            // Если блокировка не активирована, управлять
процессом
            IF NOT BlockFlag THEN
                // Определение скорости смешивания на
основе нечетких правил
                FuzzyLib.EVALUATE("IncreaseSpeed");
                FuzzyLib.EVALUATE("DecreaseSpeed");

                // Адаптация параметров нечеткой логики
на основе обратной связи
                AdaptFuzzyLogicParameters(FuzzyLib, Feed-
back);

                // Установка скорости смешивания
                SetMixerSpeed(MixerSpeed);
            ELSE
                // Аварийная остановка процесса и
отправка уведомления оператору
                ProcessRunning := FALSE;
                SetMixerSpeed(0);
                SendOperatorNotifica-
tion(DiagnosisMessage);
            END_IF;

        ELSE
            // Отправить уведомление об оперативной
блокировке
            SendOperatorNotification("Блокировка!
Необходимо решить проблему.");
        END_IF;
```

```
        // Пауза между итерациями
        WAIT(1000); // ожидание 1 с
    UNTIL FALSE;
END_PROGRAM

( Вспомогательные функции )
FUNCTION ReadProductQualitySensor: REAL;
BEGIN
    // Логика считывания данных с датчика качества
продукта
    ReadProductQualitySensor := ...;
END_FUNCTION

FUNCTION ReadTemperatureSensor: REAL;
BEGIN
    // Логика считывания данных с температурного датчика
    ReadTemperatureSensor := ...;
END_FUNCTION

FUNCTION ReadMixingTimeSensor: TIME;
BEGIN
    // Логика считывания времени смешивания
    ReadMixingTimeSensor := ...;
END_FUNCTION

FUNCTION ReadViscositySensor: REAL;
BEGIN
    // Логика считывания данных с датчика вязкости
    ReadViscositySensor := ...;
END_FUNCTION

FUNCTION ReadFeedbackSensor: REAL;
BEGIN
    // Логика считывания данных обратной связи от
процесса смешивания
    ReadFeedbackSensor := ...;
END_FUNCTION

FUNCTION AdaptFuzzyLogicParameters(FuzzyLib:
FUZZY_LIBRARY, Feedback: REAL);
BEGIN
    // Логика адаптации параметров нечеткой логики на
основе обратной связи
    // Например, можно изменять функции принадлежности
или нечеткие правила
END_FUNCTION
```

```
FUNCTION SetMixerSpeed(Speed: REAL);
BEGIN
    // Логика установки скорости смешивания
    ...
END_FUNCTION

FUNCTION SendOperatorNotification(Message: STRING);
BEGIN
    // Логика отправки уведомления оператору
    ...
END_FUNCTION
```

В этом коде происходит следующее.

1. Считывание данных с датчиков. Программа считывает значения параметров, таких как качество продукта, температура, время смешивания и вязкость, с соответствующих датчиков.

2. Проверка на соответствие стандартам. Если любое из значений параметров не соответствует установленным стандартам, активируется блокировка и отправляется сообщение оператору.

3. Управление процессом. Если блокировка не активирована, программа управляет процессом смешивания, адаптируя скорость смешивания на основе нечетких правил и обратной связи.

4. Аварийная остановка. В случае активации блокировки процесс смешивания останавливается и отправляется уведомление оператору.

Этот код обеспечивает комплексный подход к управлению процессом смешивания, включая диагностику и адаптацию параметров управления в режиме реального времени.

Этот процесс управления качеством и реагирования на возникающие проблемы в процессе смешивания имеет решающее значение для обеспечения стабильности и надежности производственных операций. Внедрение автоматизированного контроля и адаптивных механизмов позволяет оперативно выявлять и устранять не только очевидные, но и потенциальные проблемы, что значительно сокращает риски производственных сбоев и улучшает общую эффективность процессов.

С помощью интеллектуальных систем управления, способных анализировать большие объемы данных в режиме реального времени, можно оптимизировать процессы на многих уровнях. Например, алгоритмы нечеткой логики позволяют более точно реагировать на изменения в параметрах производства, адаптируясь к ним динамично, без жестко заданных порогов. Такой подход не только повышает ка-

чество конечной продукции, но и способствует экономии ресурсов за счет более точного расхода сырья и энергии.

Внедрение механизмов блокирования и аварийной остановки процесса при обнаружении критических отклонений является важной мерой предосторожности. Это не только предотвращает возможные аварии и поломки оборудования, но и гарантирует безопасность рабочего персонала, предупреждая возможные травмы или другие несчастные случаи на производстве.

Таким образом, автоматизация управления процессами не только повышает качество продукции и эффективность производства, но и является ключом к повышению безопасности и сокращению эксплуатационных расходов. Это создает основу для устойчивого развития производства и поддержания высоких стандартов качества продукции.

Формализация сложной байесовской сети может помочь в управлении технологическим процессом, включая обработку и анализ различных параметров, которые могут быть взаимосвязаны. Байесовская сеть представляет собой граф, где узлы соответствуют случайным переменным, а ребра отражают зависимости между ними. Для нашего случая можно использовать байесовскую сеть для прогнозирования качества продукта на основе различных входных параметров и для принятия управленческих решений.

Формализация байесовской сети

1. Определение переменных:

– \(Q \): качество продукта (дискретная переменная: {низкое, среднее, высокое});

– \(T \): температура смеси (непрерывная переменная);

– \(V \): вязкость продукта (непрерывная переменная);

– \(M \): время смешивания (непрерывная переменная);

– \(S \): скорость смешивания (непрерывная переменная);

– \(L \): уровень ингредиентов (дискретная переменная: {низкий, нормальный}).

2. Структура сети:

$-(T \to Q)$;

$-(V \to Q)$;

$-(M \to Q)$;

$-(S \to Q)$;

$-(L \to Q)$.

Пример кода на Python с использованием библиотеки pgmpy.

Для реализации байесовской сети мы используем библиотеку pgmpy на Python.

```python
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

Определяем структуру сети:

$$model = \\ = BayesianNetwork\left(\left[('T',\ 'Q'),('V',\ 'Q'\right),('M',\ 'Q',('S',\ 'Q',('L',\ 'Q')\right), \quad (1)$$

Определяем CPD для каждого узла:

CPD для температуры

$$cpd_T = TabularCPD\begin{pmatrix} variable = 'T', \\ variable_card = 3, \\ values = [[0.3],[0.4],[0.3]] \end{pmatrix}, \quad (2)$$

CPD для вязкости

$$cpd_V = TabularCPD\begin{pmatrix} variable = 'V', \\ variable_card = 3, \\ values = [[0.2],[0.5],[0.3]] \end{pmatrix}, \quad (3)$$

CPD для времени смешивания

$$\text{cpd\_M = TabularCPD(variable='M',} \\ \text{variable\_card=3, values=[[0.4], [0.4], [0.2]]),} \quad (4)$$

CPD для скорости смешивания

$$\text{cpd\_S = TabularCPD(variable='S', variable\_card=3,} \\ \text{values=[[0.3], [0.4], [0.3]]),} \quad (5)$$

CPD для уровня ингредиентов

$$\text{cpd\_L = TabularCPD(variable='L', variable\_card=2,} \\ \text{values=[[0.7], [0.3]]),} \quad (6)$$

CPD для качества продукта

```
cpd_Q = TabularCPD(variable='Q', variable_card=3,
values=[[0.9, 0.7, 0.8, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1,
0.3, 0.2, 0.1, 0.3, 0.2, 0.1, 0.3, 0.2, 0.1],
```

```
[0.08, 0.2, 0.15, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.3,
0.45, 0.5, 0.3, 0.4, 0.45, 0.5, 0.4, 0.35],
[0.02, 0.1, 0.05, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.4,
0.35, 0.4, 0.4, 0.4, 0.35, 0.4, 0.4, 0.35]],
evidence=['T', 'V', 'M', 'S', 'L'],
evidence_card=[3, 3, 3, 3, 2])
```

Добавляем CPD к модели:

```
model.add_cpds(cpd_T, cpd_V, cpd_M, cpd_S, cpd_L, cpd_Q)
```

Проверяем модель на корректность:

```
model.check_model()
```

Инференс:

```
inference = VariableElimination(model)
```

Пример запроса:

```
query_result = inference.query(variables=['Q'], evi-
dence={'T': 2, 'V': 1, 'M': 0, 'S': 1, 'L': 0})
print(query_result)
```
```

Обоснование использования байесовской сети.

Использование байесовской сети позволяет учитывать вероятностные зависимости между различными параметрами процесса смешивания. Это особенно полезно в ситуациях, когда данные имеют неопределенность или шум. Байесовская сеть помогает делать следующее.

1. Моделировать сложные зависимости. Взаимодействие между различными параметрами процесса смешивания, такими как температура, вязкость и скорость, может быть сложно моделировать традиционными методами. Байесовская сеть предоставляет гибкий инструмент для этого.

2. Прогнозировать качество продукции. На основе текущих параметров процесса можно прогнозировать качество конечного продукта, что позволяет своевременно принимать меры для улучшения качества.

3. Диагностировать проблемы. Сеть может выявлять вероятные причины отклонений в качестве продукции, что помогает быстрее находить и устранять проблемы.

4. Адаптивное управление. На основе предсказанных вероятностей можно динамически изменять параметры процесса для оптимизации качества продукции.
```

Таким образом, байесовская сеть является мощным инструментом для управления и оптимизации технологических процессов в условиях неопределенности (рис. 9).

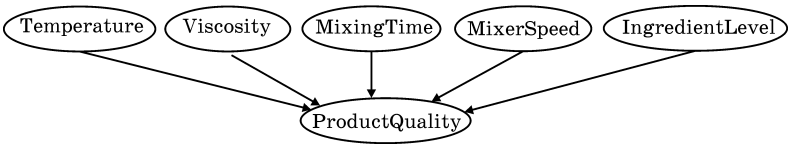

Рис. 9

Байесовская сеть, показывающая зависимости между параметрами температуры, вязкости, времени смешивания, скорости смешивания, уровня ингредиентов и качества продукта

4.4. Пошаговые инструкции по разработке и внедрению

Разработка и внедрение байесовской сети для управления технологическим процессом требуют последовательного подхода и тщательной проработки всех этапов. В первую очередь необходимо определить основные переменные, которые будут использованы в сети. Это могут быть такие параметры, как температура, вязкость, время смешивания, скорость смешивания, уровень ингредиентов и качество продукта. Каждую из этих переменных нужно четко формализовать, установив допустимые диапазоны значений и возможные состояния.

Следующим шагом является создание структуры байесовской сети. Для этого нужно определить, какие переменные влияют на качество продукта, и установить между ними причинно-следственные связи. Например, можно установить, что температура, вязкость, время смешивания, скорость смешивания и уровень ингредиентов влияют на качество продукта. Эти зависимости будут отображены в виде направленных ребер между узлами сети.

После определения структуры сети необходимо собрать и подготовить данные для оценки условных вероятностей. Данные можно получить из предыдущих записей о процессе производства, лабораторных экспериментов или симуляций. На основе этих данных нужно вычислить условные вероятности для каждой переменной, учитывая влияние других переменных.

Затем следует реализовать байесовскую сеть в программном обеспечении. Для этого можно использовать специализированные библиотеки, такие как pgmpy для Python. Реализация включает определение структуры сети, ввод данных и расчет условных вероятно-

стей. На этом этапе важно проверить корректность модели и убедиться, что все связи и зависимости отражены правильно.

После завершения разработки модели наступает этап тестирования и отладки. Байесовскую сеть нужно протестировать на реальных данных, чтобы убедиться в ее точности и надежности. В процессе тестирования можно выявить и исправить возможные ошибки, а также оптимизировать параметры модели для повышения ее эффективности.

Внедрение байесовской сети в реальный производственный процесс требует интеграции модели с системой управления процессом. Это может включать разработку интерфейсов для обмена данными между байесовской сетью и контроллерами, а также создание пользовательских интерфейсов для отображения информации и управления процессом. На этом этапе также важно обучить операторов и технический персонал работе с новой системой.

Завершив разработку и внедрение байесовской сети, необходимо организовать постоянный мониторинг и обновление модели. Это включает регулярное обновление данных, пересмотр условных вероятностей и адаптацию структуры сети по мере изменения производственного процесса. Постоянное совершенствование модели позволит поддерживать высокое качество продукции и эффективно управлять производственным процессом.

Разработка и внедрение сложной байесовской сети для управления технологическим процессом требует многослойного подхода и интеграции множества факторов. Первым этапом является тщательный анализ процесса для определения ключевых переменных, таких как температура, вязкость, время смешивания, скорость смешивания, уровень ингредиентов и качество продукта. Эти переменные необходимо формализовать, установив допустимые диапазоны значений и возможные состояния, а также учитывать потенциальные внешние факторы, такие как влажность окружающей среды и свойства сырья.

После определения переменных необходимо создать структуру байесовской сети. Это включает установление причинно-следственных связей между переменными, что позволяет моделировать сложные взаимодействия внутри процесса. Например, можно учитывать, что температура и вязкость влияют не только на качество продукта, но и друг на друга. Такие взаимозависимости отображаются в виде направленных ребер между узлами сети, что позволяет учитывать сложные многомерные зависимости.

Следующим шагом является сбор и подготовка данных для оценки условных вероятностей. Данные могут быть получены из различных источников, включая исторические записи о производстве,

лабораторные эксперименты, сенсоры в режиме реального времени и симуляции. Важно обеспечить высокое качество и репрезентативность данных, чтобы модель могла точно отражать реальность. Это также включает предварительную обработку данных для устранения пропусков и выбросов, а также нормализацию данных для унификации масштабов различных переменных.

Для реализации байесовской сети в программном обеспечении можно использовать специализированные библиотеки, такие как pgmpy для Python. Этот этап включает определение структуры сети, ввод данных и расчет условных вероятностей. Важно тщательно проверить корректность модели и убедиться, что все связи и зависимости отражены правильно. Это можно сделать с помощью различных методов валидации, включая кросс-валидацию и бутстреп.

После завершения разработки модели следует этап тестирования и отладки. Байесовскую сеть нужно протестировать на различных наборах данных, чтобы убедиться в ее точности и надежности.

Это включает проверку модели на новых данных, не использованных при обучении, чтобы оценить ее обобщающую способность. В процессе тестирования можно выявить и исправить возможные ошибки, а также оптимизировать параметры модели для повышения ее эффективности. Важно также проверить устойчивость модели к шумам и выбросам в данных, чтобы она могла адекватно работать в реальных условиях.

Внедрение байесовской сети в реальный производственный процесс требует интеграции модели с системой управления процессом. Это может включать разработку интерфейсов для обмена данными между байесовской сетью и контроллерами, а также создание пользовательских интерфейсов для отображения информации и управления процессом. Важно обеспечить надежное и своевременное получение данных с производственного оборудования и их передачу в модель для анализа. На этом этапе также необходимо обучить операторов и технический персонал работе с новой системой, что включает обучение методам интерпретации результатов байесовской сети и принятию решений на их основе.

Завершив разработку и внедрение байесовской сети, необходимо организовать постоянный мониторинг и обновление модели. Это включает регулярное обновление данных, пересмотр условных вероятностей и адаптацию структуры сети по мере изменения производственного процесса. Постоянное совершенствование модели позволит поддерживать высокое качество продукции и эффективно управлять производственным процессом. Важно также развивать систему сбора

данных и расширять базу знаний для дальнейшего улучшения модели и более точного прогнозирования результатов.

В сложных производственных процессах могут возникать ситуации, требующие оперативного вмешательства и корректировки модели [104–111]. Поэтому необходимо предусмотреть механизмы для быстрого обновления модели и ее перенастройки в случае изменения условий или выявления новых факторов, влияющих на процесс. Такой подход позволит обеспечить гибкость и адаптивность системы управления, что особенно важно в условиях динамичного и конкурентного рынка.

4.5. Обеспечение надежности и безопасности PLC-систем

Обеспечение надежности и безопасности PLC-систем является критически важным аспектом при разработке и внедрении автоматизированных систем управления. Надежность системы начинается с выбора качественного оборудования и компонентов, которые соответствуют стандартам и требованиям конкретного производственного процесса. Не менее важно обеспечить правильную установку и настройку всех устройств, чтобы минимизировать риск отказов и сбоев.

Для повышения надежности PLC-систем необходимо внедрить регулярное техническое обслуживание и мониторинг состояния оборудования. Это включает проведение профилактических осмотров, своевременную замену изношенных компонентов и использование диагностических инструментов для выявления потенциальных проблем до их возникновения. Важно также обеспечить наличие резервных копий программного обеспечения и конфигураций, чтобы быстро восстановить систему в случае сбоя.

Безопасность PLC-систем требует тщательной проработки как аппаратных, так и программных аспектов. На уровне аппаратного обеспечения необходимо предусмотреть меры защиты от несанкционированного доступа и физического повреждения устройств. Это может включать использование защищенных шкафов, контролируемых зон доступа и защитных панелей.

На уровне программного обеспечения важно реализовать механизмы аутентификации и авторизации для контроля доступа к системе. Это включает использование паролей, цифровых сертификатов и других методов аутентификации для обеспечения того, что только уполномоченные пользователи имеют доступ к критическим функциям системы. Также необходимо внедрить журналы аудита и монито-

ринга, чтобы отслеживать все действия пользователей и своевременно выявлять подозрительную активность.

Для защиты PLC-систем от киберугроз важно использовать современные средства кибербезопасности, такие как межсетевые экраны, системы обнаружения вторжений и антивирусное программное обеспечение. Регулярное обновление программного обеспечения и своевременное применение патчей безопасности также являются важными мерами для предотвращения эксплуатации уязвимостей.

Особое внимание следует уделить разработке и тестированию аварийных сценариев и процедур восстановления после сбоев. Это включает создание планов на случай аварийных ситуаций, проведение регулярных учений и тестов, а также обеспечение готовности персонала к оперативным действиям в случае возникновения чрезвычайных ситуаций. Разработка и поддержание документации по аварийным процедурам и контактной информации также важны для обеспечения готовности к быстрому реагированию.

Наконец, обеспечение надежности и безопасности PLC-систем требует постоянного обучения и повышения квалификации персонала. Операторы и технический персонал должны быть хорошо знакомы с принципами работы системы, методами диагностики и восстановления, а также современными требованиями кибербезопасности. Регулярные тренинги и курсы повышения квалификации помогут поддерживать высокий уровень компетенции и готовности к решению возникающих задач.

Таким образом, комплексный подход к обеспечению надежности и безопасности PLC-систем, включающий технические, программные и организационные меры, позволяет создать устойчивую и безопасную автоматизированную систему управления, способную эффективно функционировать в условиях современного производства.

Кейс: внедрение системы обеспечения надежности и безопасности PLC-систем

Шаг 1: анализ текущей системы и определение требований.

Первым шагом в обеспечении надежности и безопасности PLC-систем является проведение всестороннего анализа текущей системы. Это включает в себя оценку состояния оборудования, программного обеспечения, сетевой инфраструктуры и процедур безопасности. Необходимо определить ключевые уязвимости и риски, которые могут повлиять на работу системы. На этом этапе также важно собрать требования к надежности и безопасности от всех заинтересованных сторон, включая операционный персонал, технических специалистов и руководителей предприятия.

Шаг 2: выбор и установка качественного оборудования.

На основе анализа необходимо выбрать оборудование, которое соответствует современным стандартам надежности и безопасности. Это включает в себя использование контроллеров и периферийных устройств, сертифицированных для промышленного использования, с учетом их совместимости и функциональных возможностей. Установка оборудования должна быть выполнена в соответствии с рекомендациями производителей и с учетом всех технических и эксплуатационных требований.

Шаг 3: настройка и конфигурация системы.

После установки оборудования необходимо выполнить его настройку и конфигурацию. Это включает в себя программирование контроллеров, настройку сетевых параметров и конфигурацию периферийных устройств. На этом этапе также важно реализовать базовые меры безопасности, такие как аутентификация и авторизация пользователей, ограничение доступа к критическим компонентам системы и установка межсетевых экранов.

Шаг 4: внедрение процедур технического обслуживания и мониторинга.

Для поддержания надежности системы необходимо внедрить регулярные процедуры технического обслуживания и мониторинга. Это включает в себя проведение плановых профилактических осмотров, диагностику состояния оборудования с использованием специализированных инструментов и систем, а также своевременную замену изношенных компонентов. Важно также настроить систему мониторинга, которая будет отслеживать ключевые параметры работы системы и уведомлять о возможных проблемах.

Шаг 5: реализация кибербезопасности.

Кибербезопасность является неотъемлемой частью обеспечения надежности PLC-систем. Необходимо внедрить современные средства кибербезопасности, такие как системы обнаружения вторжений, антивирусное программное обеспечение и межсетевые экраны. Важно регулярно обновлять программное обеспечение контроллеров и периферийных устройств, своевременно устанавливая патчи безопасности для устранения выявленных уязвимостей.

Шаг 6: разработка и тестирование аварийных сценариев.

Для обеспечения готовности к чрезвычайным ситуациям необходимо разработать и протестировать аварийные сценарии. Это включает создание планов на случай аварийных ситуаций, проведение регулярных учений и тестов, а также обучение персонала действиям в условиях аварий. Важно создать документацию, включающую процедуры

аварийного реагирования и контактную информацию, чтобы персонал был готов к быстрому реагированию.

Шаг 7: обучение и повышение квалификации персонала.

Операторы и технический персонал должны быть хорошо обучены и иметь высокую квалификацию для работы с PLC-системами. Регулярные тренинги и курсы повышения квалификации помогут поддерживать уровень компетенции и готовности к решению возникающих задач. Обучение должно охватывать как технические аспекты работы с оборудованием, так и методы обеспечения безопасности и диагностики системы.

Шаг 8: постоянный мониторинг и обновление системы.

После внедрения всех мер по обеспечению надежности и безопасности необходимо организовать постоянный мониторинг и регулярное обновление системы. Это включает в себя непрерывный сбор данных о состоянии системы, пересмотр и обновление процедур безопасности, а также адаптацию системы к изменяющимся условиям и требованиям производства. Постоянное совершенствование системы позволит поддерживать высокий уровень надежности и безопасности в долгосрочной перспективе.

Пример реализации на практике.

Для иллюстрации процесса внедрения рассмотрим предприятие, занимающееся смешиванием химических веществ. На этом предприятии было решено модернизировать существующую PLC-систему для повышения ее надежности и безопасности.

1. Анализ текущей системы. Был проведен аудит существующей PLC-системы, выявлены устаревшие компоненты и уязвимости в программном обеспечении. Определены требования к обновленной системе, включая повышение устойчивости к внешним воздействиям и киберугрозам.

2. Выбор оборудования. Были выбраны современные контроллеры с поддержкой передовых протоколов безопасности и возможности удаленного мониторинга. Также были обновлены сенсоры и исполнительные механизмы для повышения точности и надежности системы.

3. Настройка и конфигурация. В процессе настройки были реализованы меры безопасности, включая многофакторную аутентификацию для доступа к системе управления и настройку межсетевых экранов для защиты от внешних угроз. Были установлены системы резервного копирования и восстановления данных.

4. Внедрение технического обслуживания. Разработан план регулярного технического обслуживания, включающий диагностику

состояния оборудования и профилактические осмотры. Были установлены системы мониторинга, отслеживающие ключевые параметры работы системы и уведомляющие о возможных проблемах.

5. Реализация кибербезопасности. Внедрены системы обнаружения вторжений и антивирусное программное обеспечение. Настроены автоматические обновления для своевременного применения патчей безопасности.

6. Аварийные сценарии. Разработаны и протестированы сценарии на случай аварийных ситуаций. Проведены тренировки персонала, включающие действия в условиях аварий.

7. Обучение персонала. Организованы регулярные тренинги для операторов и технического персонала, охватывающие как технические аспекты работы с оборудованием, так и методы обеспечения безопасности.

8. Постоянный мониторинг. Установлены системы постоянного мониторинга состояния системы, организован сбор данных для анализа и регулярное обновление процедур безопасности и технического обслуживания.

Этот пример иллюстрирует комплексный подход к обеспечению надежности и безопасности PLC-систем, включающий технические, программные и организационные меры. Такой подход позволяет создать устойчивую и безопасную систему управления, способную эффективно функционировать в условиях современного производства.

Для создания схемы развертывания байесовской сети в виде дерева решений рассмотрим процесс развертывания этой сети в контексте конкретного применения. В этом примере мы будем использовать дерево решений для принятия решений на основе параметров технологического процесса смешивания.

Шаг 1: определение узлов дерева решений.

В нашем дереве решений будут следующие узлы:

– температура (Temperature);

– вязкость (Viscosity);

– время смешивания (Mixing Time);

– скорость смешивания (Mixer Speed);

– уровень ингредиентов (Ingredient Level);

– качество продукта (Product Quality).

Дерево решений будет начинаться с узла, который имеет наибольшее влияние на качество продукта, и разветвляться в зависимости от значений параметров. В нашем примере начнем с температуры (рис. 10).

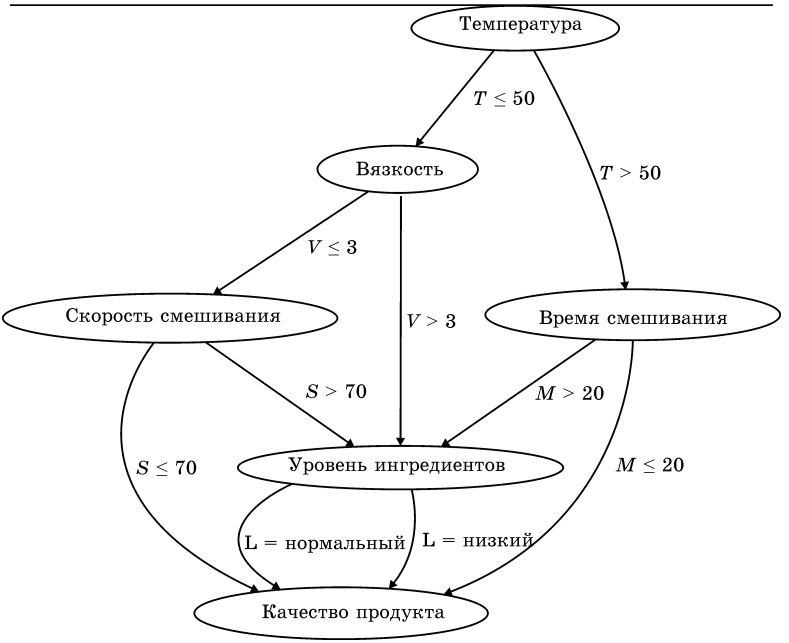

Рис. 10
Структура дерева решений

Формализация в псевдокоде.

Для формализации дерева решений в псевдокоде мы можем представить его в виде логических условий и ветвлений. Псевдокод будет включать последовательность проверок параметров процесса и соответствующих действий для обеспечения качества продукта.

Псевдокод для дерева решений:

```pseudo
function evaluateProcess(temperature, viscosity, mixing-
Time, mixerSpeed, ingredientLevel):
    if temperature <= 50:
        if viscosity <= 3:
            if mixerSpeed <= 70:
                if ingredientLevel == "нормальный":
                    return "Качество продукта: высокое"
                else:
                    return "Качество продукта: среднее"
            else:
                if ingredientLevel == "нормальный":
                    return "Качество продукта: среднее"
```

```
                else:
                    return "Качество продукта: низкое"
        else:
            if ingredientLevel == "нормальный":
                return "Качество продукта: среднее"
            else:
                return "Качество продукта: низкое"
    else:
        if mixingTime <= 20:
            if ingredientLevel == "нормальный":
                return "Качество продукта: высокое"
            else:
                return "Качество продукта: среднее"
        else:
            if ingredientLevel == "нормальный":
                return "Качество продукта: среднее"
            else:
                return "Качество продукта: низкое"
```

Пример вызова функции:

```
temperature = 55
viscosity = 4
mixingTime = 15
mixerSpeed = 60
ingredientLevel = "нормальный"

result = evaluateProcess(temperature, viscosity, mixing-
Time, mixerSpeed, ingredientLevel)
print(result)
```

Обоснование структуры псевдокода.

1. Температура. Первый условный оператор проверяет, находится ли температура ниже или выше порогового значения 50. Температура имеет значительное влияние на вязкость и смешивание компонентов, что определяет дальнейшие проверки.

2. Вязкость. Внутри ветки, соответствующей температуре ниже 50, проверяется вязкость. Это важный параметр, влияющий на смешивание компонентов.

3. Скорость смешивания. Если вязкость ниже или равна 3, проверяется скорость смешивания. Это позволяет уточнить, как скорость влияет на процесс.

4. Уровень ингредиентов. В каждой из ветвей проверяется уровень ингредиентов, чтобы определить его влияние на качество продукта.

5. Время смешивания. Если температура выше 50, проверяется время смешивания. Это важный параметр, влияющий на степень однородности смеси.

6. Финальные условия. В каждой ветви проверяется уровень ингредиентов и на основе всех предыдущих проверок определяется качество продукта.

Пример вызова функции.

Для тестирования данной логики можно вызвать функцию evaluateProcess с различными значениями параметров, как показано в примере. Это позволит оценить, как разные условия влияют на конечное качество продукта.

Этот псевдокод позволяет формализовать дерево решений для процесса смешивания, учитывая все ключевые параметры и их влияние на качество продукта. Его можно использовать для разработки более сложных систем управления и принятия решений на основе данных.

Этот псевдокод позволяет формализовать дерево решений для процесса смешивания, учитывая все ключевые параметры и их влияние на качество продукта. Его можно использовать для разработки более сложных систем управления и принятия решений на основе данных. Например, в условиях химического производства, где требуется строгий контроль качества продукции, дерево решений может служить основой для автоматизированной системы контроля качества.

Рассмотрим более сложный пример использования дерева решений в химическом производстве. Допустим, мы производим химическое вещество, требующее точного контроля за температурой, вязкостью, временем и скоростью смешивания, а также уровнем ингредиентов. Качество конечного продукта сильно зависит от этих параметров, и отклонение от оптимальных значений может привести к браку.

Пример более сложной системы:

```
function evaluateChemicalProcess(temperature, viscosity,
mixingTime, mixerSpeed, ingredientLevel, pH, concentra-
tion):
    if temperature <= 50:
        if viscosity <= 3:
            if pH < 7:
                if concentration > 10:
                    if mixerSpeed <= 70:
                        if ingredientLevel ==
"нормальный":
                            return "Качество продукта:
высокое"
```

```
                        else:
                            return "Качество продукта:
среднее"
                    else:
                        if ingredientLevel ==
"нормальный":
                            return "Качество продукта:
среднее"
                        else:
                            return "Качество продукта:
низкое"
                else:
                    if ingredientLevel == "нормальный":
                        return "Качество продукта:
среднее"
                    else:
                        return "Качество продукта:
низкое"
            else:
                if concentration > 10:
                    if mixerSpeed <= 70:
                        if ingredientLevel ==
"нормальный":
                            return "Качество продукта:
среднее"
                        else:
                            return "Качество продукта:
низкое"
                    else:
                        if ingredientLevel ==
"нормальный":
                            return "Качество продукта:
низкое"
                        else:
                            return "Качество продукта:
очень низкое"
                else:
                    if ingredientLevel == "нормальный":
                        return "Качество продукта:
низкое"
                    else:
                        return "Качество продукта: очень
низкое"
        else:
            if pH < 7:
                if concentration > 10:
                    return "Качество продукта: среднее"
```

```
                else:
                    return "Качество продукта: низкое"
            else:
                if concentration > 10:
                    return "Качество продукта: низкое"
                else:
                    return "Качество продукта: очень
низкое"
    else:
        if mixingTime <= 20:
            if ingredientLevel == "нормальный":
                return "Качество продукта: высокое"
            else:
                return "Качество продукта: среднее"
        else:
            if ingredientLevel == "нормальный":
                return "Качество продукта: среднее"
            else:
                return "Качество продукта: низкое"
```

Пример вызова функции:

```
temperature = 55
viscosity = 4
mixingTime = 15
mixerSpeed = 60
ingredientLevel = "нормальный"
pH = 6.5
concentration = 12

result = evaluateChemicalProcess(temperature, viscosity,
mixingTime, mixerSpeed, ingredientLevel, pH, concentra-
tion)
print(result)
```

Обоснование использования дерева решений.

В данном примере псевдокод учитывает дополнительные параметры, такие как уровень pH и концентрация химических веществ, которые также могут существенно влиять на качество конечного продукта. Дерево решений позволяет автоматизированной системе контроля принимать обоснованные решения на основе текущих параметров процесса, помогая поддерживать высокий уровень качества продукции.

Применение в реальном производстве.

В реальном химическом производстве такая система может быть интегрирована с сенсорами и контроллерами, которые постоянно мониторят параметры процесса и автоматически регулируют их, чтобы обес-

печить оптимальные условия. Дерево решений в сочетании с байесовской сетью может обеспечить более точное и гибкое управление, позволяя быстро адаптироваться к изменениям в производственных условиях.

Использование деревьев решений и других методов искусственного интеллекта в промышленной автоматизации открывает новые возможности для повышения эффективности и качества производственных процессов. Они позволяют не только автоматизировать рутинные задачи, но и принимать сложные решения, основанные на анализе данных, что в конечном итоге приводит к улучшению качества продукции и снижению затрат.

ЗАКЛЮЧЕНИЕ

В процессе изучения и внедрения систем программируемых логических контроллеров (PLC) в промышленных автоматизированных процессах были рассмотрены различные аспекты, начиная от основ программирования до сложных методологий управления и обеспечения безопасности. PLC-системы играют ключевую роль в современных производственных и технологических процессах, обеспечивая надежное и эффективное управление.

Одним из важнейших этапов в освоении PLC является понимание основ программирования и конфигурации контроллеров. В первой части учебника были подробно рассмотрены типы логических контроллеров, их архитектура, принципы работы и функциональные возможности. Особое внимание уделено различным языкам программирования PLC, таким как Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) и Sequential Function Chart (SFC). Эти языки позволяют гибко и эффективно описывать логические процессы и алгоритмы управления, обеспечивая высокую точность и надежность выполнения задач.

В дальнейшем особое внимание уделялось методологиям структурного и модульного программирования, что позволяет создавать масштабируемые и легко поддерживаемые системы. Примеры и кейсы из реальной практики продемонстрировали, как применение этих методологий помогает решать сложные задачи автоматизации и оптимизации производственных процессов.

Отладка и тестирование программ для PLC являются неотъемлемой частью разработки. В учебнике рассмотрены различные методы и инструменты, которые помогают выявлять и устранять ошибки на ранних стадиях разработки, что значительно снижает риск возникновения сбоев в реальной эксплуатации. Особое внимание уделено симуляции процессов, которая позволяет протестировать логику управления без необходимости остановки реального производства.

Интеграция сложных алгоритмов, таких как нечеткие логические алгоритмы и байесовские сети, позволяет создавать более адаптивные и интеллектуальные системы управления. В учебнике приведены примеры использования этих методов для оптимизации технологических процессов, что позволяет повысить качество продукции и снизить затраты на производство. Байесовские сети, в частности, продемонстрировали свою эффективность в моделировании сложных зависимостей и прогнозировании исходов на основе многомерных данных.

Безопасность и надежность PLC-систем являются критически важными аспектами, особенно в условиях современной промышленности, где любая ошибка может привести к серьезным последствиям. В учебнике подробно описаны методы обеспечения безопасности, включая физическую защиту оборудования, кибербезопасность, а также разработку и тестирование аварийных сценариев. Внедрение систем мониторинга и регулярного технического обслуживания позволяет поддерживать высокую надежность и готовность оборудования.

В заключение следует отметить, что изучение программирования логических контроллеров и их применение в промышленной автоматизации позволяет решать широкий спектр задач, обеспечивая высокую эффективность, надежность и безопасность производственных процессов. Этот учебник представляет собой всестороннее руководство, охватывающее как базовые, так и продвинутые темы и является ценным ресурсом для инженеров и специалистов по автоматизации.

Применение описанных методов и технологий на практике позволит создавать высокотехнологичные и конкурентоспособные производственные системы, отвечающие современным требованиям и вызовам промышленности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Научная, учебная и учебно-методическая литература, периодические издания

1. *Cooksy, L. J.* The program logic model as an integrative framework for a multimethod evaluation / L. J. Cooksy, P. Gill, P. A. Kelly // Evaluation and program planning. — 2001. — Т. 24, № 2. — P. 119–128.

2. *Ahmed, I.* Programmable logic controller forensics / I. Ahmed [et al.] // IEEE Security & Privacy. — 2017. — Т. 15, № 6. — P. 18–24.

3. *Hudedmani, M. G.* Programmable logic controller (PLC) in automation / M. G. Hudedmani [et al.] // Advanced Journal of Graduate Research. — 2017. — Т. 2, № 1. — P. 37–45.

4. *Foster, M.* A review of Programmable Logic Controllers in control systems education / M. Foster, C. Hammerquist, R. Melendy // 2010 Annual Conference & Exposition. — 2010. — P. 15.82. 1–15.82. 10.

5. *Burhan, I.* Multiple input/outputs Programmable Logic Controller (PLC) module for educational applications / I. Burhan, A. A. Azman, S. Talib // 2015 Innovation & Commercialization of Medical Electronic Technology Conference (ICMET). — IEEE, 2015. — P. 39–43.

6. *Sehr, M. A.* Programmable logic controllers in the context of industry 4.0 / M. A. Sehr [et al.] // IEEE Transactions on Industrial Informatics. — 2020. — Т. 17, № 5. — P. 3523–3533.

7. *Hajarnavis, V.* An investigation into programmable logic controller software design techniques in the automotive industry / V. Hajarnavis, K. Young // Assembly Automation. — 2008. — Т. 28, № 1. — P. 43–54.

8. *Qasim, S. A.* Automated reconstruction of control logic for programmable logic controller forensics / S. A. Qasim, J. Lopez, I. Ahmed // Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16–18, 2019, Proceedings 22. — Springer International Publishing, 2019. — P. 402–422.

9. *Yau, K.* Detecting anomalous programmable logic controller events using machine learning / K. Yau, K. P. Chow // Advances in Digital Forensics XIII: 13th IFIP WG 11.9 International Conference, Orlando, FL, USA, January 30 — February 1, 2017, Revised Selected Papers 13. — Springer International Publishing, 2017. — P. 81–94.

10. *Yau, K.* PLC forensics based on control program logic change detection / K. Yau, K. P. Chow // Journal of Digital Forensics, Security and Law. — 2015. — Т. 10, № 4. — P. 5.

11. *Akparibo, A. R.* Development of a programmable logic controller training platform for the industrial control of processes / A. R. Akparibo

[et al.] // American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS). — 2016. — Т. 15, № 1. — P. 186–196.

12. *Serhane, A.* Programmable logic controllers based systems (PLC-BS): Vulnerabilities and threats / A. Serhane [et al.] // SN Applied Sciences. — 2019. — Т. 1. — P. 1–12.

13. *Martín-Liras, L.* Comparative analysis of the security of configuration protocols for industrial control devices / L. Martín-Liras [et al.] // International Journal of Critical Infrastructure Protection. — 2017. — Т. 19. — P. 4–15.

14. *Hajarnavis, V.* An investigation into programmable logic controller software design techniques in the automotive industry / V. Hajarnavis, K. Young // Assembly Automation. — 2008. — Т. 28, № 1. — P. 43–54.

15. *Wang, Z.* A Survey on Programmable Logic Controller Vulnerabilities, Attacks, Detections, and Forensics / Z. Wang [et al.] // Processes. — 2023. — Т. 11, № 3. — P. 918.

16. *Cook, M. M.* Plcprint: Fingerprinting memory attacks in programmable logic controllers / M. M. Cook, A. K. Marnerides, D. Pezaros // IEEE Transactions on Information Forensics and Security. — 2023.

17. *Fu, M.* Multicrane visual sorting system based on deep learning with virtualized programmable logic controllers in industrial Internet / M. Fu [et al.] // IEEE Transactions on Industrial Informatics. — 2023.

18. *Kibulungu, J. W.* Automatic control system based on Industry 4.0, PLC, and SCADA / J. W. Kibulungu, O. T. Laseinde // Intelligent Sustainable Systems: Selected Papers of WorldS4 2022. — Singapore : Springer Nature Singapore, 2023. — Vol. 1. — P. 183–197.

19. *Cai, Y.* A review of in-situ monitoring and process control system in metal-based laser additive manufacturing / Y. Cai [et al.] // Journal of Manufacturing Systems. — 2023. — Т. 70. — P. 309–326.

20. *Oluwole-ojo, O.* Energy consumption analysis of a continuous flow ohmic heater with advanced process controls / O. Oluwole-ojo [et al.] // Energies. — 2023. — Т. 16, № 2. — P. 868.

21. *Kim, H. S.* GRU-based Buzzer Ensemble for Abnormal Detection in Industrial Control Systems / H. S. Kim [et al.] // Computers, Materials & Continua. — 2023. — Т. 74, № 1.

22. *Tomar, I.* Real Time Control System for Metro Railways Using PLC & SCADA / I. Tomar, I. Sreedevi, N. Pandey // Intelligent Automation & Soft Computing. — 2023. — Т. 35, № 2.

23. *Minhat, M. S.* et al. Hybrid MPC-P controller for the core power control system at TRIGA reactor / M. S. Minhat [et al.] // Journal of Process Control. — 2023. — Т. 122. — P. 184–198.

24. *Shezan, S. A.* Optimization and control of solar-wind islanded hybrid microgrid by using heuristic and deterministic optimization algorithms and fuzzy logic controller / S. A. Shezan [et al.] // Energy reports. — 2023. — Т. 10. —— P. 3272–3288.

25. *Sun, M.* Design of intelligent manufacturing system based on digital twin for smart shop floors / M. Sun, Z. Cai, N. Zhao // International Journal of Computer Integrated Manufacturing. — 2023. — Т. 36, № 4. — P. 542–566.

26. *Lee, A.* Assessment of the Distributed Ledger Technology for Energy Sector Industrial and Operational Applications Using the MITRE ATT&CK® ICS Matrix / A. Lee [et al.] // IEEE Access. — 2023.

27. *Yu, Y.* Process stability control of corner structures in robotic gas tungsten arc additive manufacturing via arc sensing / Y. Yu [et al.] // Journal of Manufacturing Processes. — 2023. — Т. 101. — P. 156–170.

28. *Mitov, A.* Rapid Prototyping of H∞ Algorithm for Real-Time Displacement Volume Control of Axial Piston Pumps / A. Mitov, T. Slavov, J. Kralev // Algorithms. — 2023. — Т. 16, № 2. — P. 120.

29. *Chehaitly, M.* Error Correcting Codes Analysis for Wireless Industrial Communication based on Wavelet Packet Architecture / M. Chehaitly [et al.]. — 2024.

30. *Prastiyo, D.* Irrigation Sluice Control System Using Algorithm Based DC Motor PID And Omron PLC / D. Prastiyo, W. S. Aji // Control Systems and Optimization Letters. — 2023. — Т. 1, № 1. — P. 19–26.

31. *Salvador, L. C. R.* SCADA systems: Security concerns and countermeasures / L. C. R. Salvador, N. H. P. Dai, R. Zoltán // 2023 IEEE 21st World Symposium on Applied Machine Intelligence and Informatics (SAMI). — IEEE, 2023. — P. 251–254.

32. *Feng, T.* Security Analysis and Enhancement of INTERBUS Protocol in ICS Based on Colored Petri Net / T. Feng [et al.] // Information. — 2023. — Т. 14, № 11. — P. 589.

33. *Parsafar, P.* Real-time fault diagnosis system for electrical panel using embedded systems / P. Parsafar, P. Qaderi Baban // Journal of Electrical Systems and Information Technology. — 2023. — Т. 10, № 1. — P. 37.

34. *Wen, H.* Risk assessment of human-automation conflict under cyberattacks in process systems / H. Wen [et al.] // Computers & Chemical Engineering. — 2023. — Т. 172. — P. 108–175.

35. *Parant, A.* Model-based engineering for designing cyber-physical systems from product specifications / A. Parant [et al.] // Computers in Industry. — 2023. — Т. 145. — P. 103–808.

36. *Abdolrasol, M. G. M.* Optimal fuzzy logic controller based PSO for photovoltaic system / M. G. M. Abdolrasol [et al.] // Energy Reports. — 2023. — T. 9. — P. 427–434.

37. *Rodriguez, M.* Fuzzy logic-based energy management for isolated microgrid using meta-heuristic optimization algorithms / M. Rodriguez, D. Arcos-Aviles, W. Martinez // Applied Energy. — 2023. — T. 335. — P. 120–771.

38. *Bingül, Ö.* Fuzzy logic and proportional integral derivative based multi-objective optimization of active suspension system of a 4×4 in-wheel motor driven electrical vehicle / Ö. Bingül, A. Yıldız // Journal of Vibration and Control. — 2023. — T. 29, № 5–6. — P. 1366–1386.

39. *Hentout, A.* A review of the literature on fuzzy-logic approaches for collision-free path planning of manipulator robots / A. Hentout, A. Maoudj, M. Aouache // Artificial Intelligence Review. — 2023. — T. 56, № 4. — P. 3369–3444.

40. *Koay, A. M. Y.* Machine learning in industrial control system (ICS) security: current landscape, opportunities and / A. M. Y. Koay [et al.] // Journal of Intelligent Information Systems. — 2023. — T. 60, № 2. — P. 377–405.

41. *Hamieh, M.* Programming CAR T cell tumor recognition: tuned antigen sensing and logic gating / M. Hamieh [et al.] // Cancer Discovery. — 2023. — T. 13, № 4. — P. 829–843.

42. *Aguila-Leon, J.* Solar photovoltaic Maximum Power Point Tracking controller optimization using Grey Wolf Optimizer: A performance comparison between bio-inspired and traditional algorithms / J. Aguila-Leon [et al.] // Expert Systems with Applications. — 2023. — T. 211. — P. 118–700.

43. *Singh, K.* Tidal turbine support in microgrid frequency regulation through novel cascade Fuzzy-FOPID droop in de-loaded region / K. Singh, Y. Arya // ISA transactions. — 2023. — T. 133. — P. 218–232.

44. *Ryzhova, K.* Artificial intelligence in the diagnosis of diseases of various origins / K. Ryzhova, A. V. Yumashev, M. Klimova [et al.] // Journal of Complementary Medicine Research. — 2023. — Vol. 14, № 2. — P. 199–202. — DOI: 10.5455/jcmr.2023.14.02.31.

45. *Чирков, М.* Знания и информация как синергия платформенного подхода цифровизации глобального развития / М. Чирков, Т. Лачинина, М. Чистяков // Свободная мысль. — 2020. — № 5 (1683). — С. 37–44. — DOI 10.24411/0869-4435-2020-00003.

46. *Лачинина, Т. А.* Региональное развитие через усовершенствование реализации муниципальных функций в электронном виде посредством разработки административных регламентов / Т. А. Лачи-

нина, М. Ю. Казаков, М. С. Чистяков // Вопросы управления. — 2017. — № 1 (44). — С. 48–58.

47. Применение методов аналитики больших данных в макроэкономическом моделировании / Н. Н. Лытнев, А. Л. Золкин, Н. В. Левошич, С. А. Жильцов // Журнал прикладных исследований. — 2023. — № 2. — С. 73–82.

48. Системы динамического моделирования в экономике / Н. М. Нилова, А. Л. Золкин, Г. Р. Темижева, А. И. Пахомова // Журнал прикладных исследований. — 2023. — № 1. — С. 36–43.

49. Конвергенция информационной и экономической безопасности для повышения конкурентоспособности производственно-торговых предприятий на основе системы сбалансированных показателей : монография / В. В. Филатов, Н. В. Артемьев, В. В. Безпалов [и др.]. — Курск : Закрытое акционерное общество «Университетская книга», 2023. — 763 с.

50. Применение искусственного интеллекта в логистике : монография / Н. М. Нилова, А. Л. Золкин, А. И. Пахомова, А. В. Буракова. — Краснодар : Новация, 2023. — 169 с.

51. Роль и значимость систем поддержки принятия решений в современном бизнесе / В. А. Мирончук, А. Л. Золкин, Н. В. Левошич, А. Б. Урусова // Экономика и предпринимательство. — 2023. — № 9 (158). — С. 975–979.

52. Состояние рынка Business Intelligence как инструмента анализа и обработки данных : монография / С. Н. Косников, А. Л. Золкин, О. В. Сараджева, А. Б. Урусова. — Краснодар : Новация, 2023. — 161 с.

53. Методы и инструменты бизнес-аналитики для корпоративных информационно-аналитических систем : монография / С. Н. Косников, А. Л. Золкин, М. С. Чистяков, Т. Б. Матвиевская. — Краснодар : Новация, 2023. — 170 с.

54. Эффективность облачных решений в контексте систем поддержки принятия решений / В. А. Мирончук, А. Л. Золкин, Л. Б. Атаева, Ю. В. Скибин // Экономика и предпринимательство. — 2023. — № 8 (157). — С. 1466–1470.

55. Внедрение средств операционной аналитики в телекоммуникационное обеспечение передачи информации в сложной бизнес-системе единого центра хранения данных / А. Л. Золкин, Н. Ю. Логунова, Е. А. Арнаутов, А. Н. Лосев // Научно-технический вестник Поволжья. — 2023. — № 10. — С. 112–118.

56. *Косникова, О. В*. Разработка математических и программных средств моделирования в экономике / О. В. Косникова, А. Л. Золкин,

А. И. Аджиева // Естественно-гуманитарные исследования. — 2023. — № 3 (47). — С. 137–141.

57. Компьютерные системы поддержки управленческих и организационных решений / В. А. Мирончук, Т. Г. Айгумов, А. Л. Золкин, А. Б. Урусова // Естественно-гуманитарные исследования. — 2023. — № 3 (47). — С. 441–445.

58. Современные компьютерные системы поддержки принятия решений / В. А. Мирончук, А. Л. Золкин, Ж. В. Мекшенева, И. А. Поскряков // Естественно-гуманитарные исследования. — 2023. — № 4 (48). — С. 228–231.

59. *Золкин, А. Л.* Проектирование цифровых экосистем окружающего интеллекта, сенсорных и компьютерных сетей : монография / А. Л. Золкин, В. Д. Мунистер . — М. : Русайнс, 2022. — 148 р.

60. Problems of personal data and information protection in corporate computer networks / A. L. Zolkin, E. M. Abdulmukminova, V. N. Malikov, A. N. Lepshokova // IOP Conference Series : materials Science and Engineering / Krasnoyarsk Science and Technology City Hall. — Krasnoyarsk, Russian Federation, 2021. — P. 12102.

61. Research of problems of computer networks expert systems / A. L. Zolkin, A. N. Losev, D. V. Gridina, T. G. Aygumov // IOP Conference Series : materials Science and Engineering / Krasnoyarsk Science and Technology City Hall. — Krasnoyarsk, Russian Federation, 2021. — P. 12106.

62. Introduction of fog computations to measuring system collection and accounting of data of distributed Internet of Things / V. D. Munister, A. L. Zolkin, T. G. Aygumov, V. E. Ozhiganov // Journal of Physics : Conference Series. — Ser. “International Conference on Automatics and Energy, ICAE 2021”. — 2021. — P. 012155.

63. Creation of a software and hardware product of a real-time system for collecting, accounting and managing data transmission of an intelligent transport system in context of the IOT / A. L. Zolkin, V. D. Munister, E. A. Lavrov [et al.] // Journal of Physics : Conference Series / Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations. — Krasnoyarsk, Russia, 2021. — P. 52059.

64. Use of information technologies in economy with a purpose of digital improvement / A. L. Zolkin, T. G. Aygumov, V. V. Bobkov, O. V. Kosnikova // European Proceedings of Social and Behavioural Sciences EpSBS. — Krasnoyarsk, Russia, 2021. — P. 1652–1658.

65. Системный подход в моделировании эффективности кадрового потенциала сотрудников IT-предприятия / А. Л. Золкин, В. С. Тор-

мозов, Т. Н. Буштрук, Е. А. Арнаутов // Вестник Российского нового университета. — Сер. «Сложные системы: модели, анализ и управление». — 2023. — № 1. — С. 3–11.

66. Политика информационной безопасности в цифровом здравоохранении: организационно-правовые аспекты / А. Л. Марухленко, А. В. Чешин, С. С. Алеева [и др.] // Вопросы политологии. — 2023. — Т. 13, № 12 (100). — С. 6612–6624.

67. Разработка пользовательского интерфейса и управление информацией в системах поддержки принятия решений / С. Н. Косников, А. Л. Золкин, Л. Б. Атаева, С. А. Жильцов // Естественно-гуманитарные исследования. — 2023. — № 5 (49). — С. 406–409.

68. Особенности экспертных систем поддержки принятия решений и их применение в экономике / С. Н. Косников, А. Л. Золкин, Л. Б. Атаева, В. А. Доржеева // Естественно-гуманитарные исследования. — 2023. — № 5 (49). — С. 160–163.

69. Интеграция больших данных и аналитических возможностей в современные системы поддержки принятия решений / В. А. Мирончук, А. Л. Золкин, А. В. Батищев, А. Б. Урусова // Вестник Академии знаний. — 2023. — № 5 (58). — С. 227–230.

70. Анализ возможностей применения искусственного интеллекта в системах поддержки принятия решений / С. Н. Косников, А. Л. Золкин, А. В. Батищев, Д. Е. Салыкова // Вестник Академии знаний. — 2023. — № 5 (58). — С. 165–168.

71. Влияние современных технологий на эффективность систем поддержки принятия решений / И. И. Василенко, А. Л. Золкин, Т. Г. Гарбузова, Л. Б. Атаева // Экономика и предпринимательство. — 2023. — № 10 (159). — С. 1366–1371.

72. Проблемы и перспективы развития систем поддержки принятия решений в будущем / И. И. Василенко, А. Л. Золкин, Г. Р. Темижева, Т. Б. Матвиевская // Экономика и предпринимательство. — 2023. — № 10 (159). — С. 827–832.

73. *Верещагина, Е. А.* Исследование проблем информационной безопасности в банковской сфере : монография / Е. А. Верещагина, А. Л. Золкин, А. В. Фролов. — М. : Русайнс, 2023. — 178 с.

74. *Ратушняк, Г. Я.* Базы данных : учебное пособие / Г. Я. Ратушняк, А. Л. Золкин, А. Л. Никитин. — М. : Русайнс, 2022. — 128 с.

75. *Ратушняк, Г. Я.* Технологии разработки и проектирования информационных систем : учебное пособие / Г. Я. Ратушняк, А. Л. Золкин. — М. : Русайнс, 2022. — Ч. 1. — 202 с.

76. *Ратушняк, Г. Я.* Технологии разработки и проектирования информационных систем : учебное пособие / Г. Я. Ратушняк, А. Л. Золкин. — М. : Русайнс, 2022. — Ч. 2. — 350 с.

77. Применение смешанного подхода для синтеза цифровых экосистем машинного обучения / А. Л. Золкин, Т. Г. Айгумов, В. С. Тормозов, Ю. В. Гуменникова // Вестник Российского нового университета. Сер. «Сложные системы: модели, анализ и управление». — 2023. — № 1. — С. 12–19.

78. Application of methods of applied mathematics in the gateway of information exchange for business applications queuing systems / A. L. Zolkin, O. P. Shevchenko , A. N. Losev [et al.] // AIP Conference Proceedings. 2nd International Scientific Forum on Computer and Energy Sciences, WFCES-II 2021. — 2022. — P. 20023.

79. *Кнышов, А. В.* Бизнес-анализ в управлении : монография / А. В. Кнышов, А. Л. Золкин. — М. : Русайнс, 2022. — 86 с.

80. Features of the synthesis of information and measurement systems using machine learning for conducting of environmental monitoring / Yu. M. Avdeev, A. I. Pakhomova, A. L. Zolkin [et al.] // Journal of Physics : Conference Series. II International Scientific Conference on Metrological Support of Innovative Technologies (ICMSIT II-2021). — Krasnoyarsk, 2021. — P. 32008.

81. *Верещагина, Е. А.* Исследование проблем безопасности USB-интерфейсов : монография / Е. А. Верещагина, А. Л. Золкин, К. А. Василенко. — М. : Русайнс, 2024. — 154 с.

82. Интерактивные дашборды и их роль в управленческом учете / Н. Н. Лытнев, А. Л. Золкин, М. В. Бузулуцкая, Ю. В. Шмойлова // Экономика и предпринимательство. — 2024. — № 1 (162). — С. 1201–1205.

83. Большие данные как ключ к эффективному управлению рисками / О. В. Косникова, А. Л. Золкин, Н. В. Левошич, Н. Ю. Логунова // Экономика и предпринимательство. — 2024. — № 1 (162). — С. 1193–1197.

84. Трансформация мировой экономики под воздействием космических технологий / М. А. Щегринец, А. Л. Золкин, С. В. Шамина, Г. В. Рябкова // Экономика и предпринимательство. — 2024. — № 1 (162). — С. 404–407.

85. Влияние демографических сдвигов на экономику Российской Федерации / М. А. Щегринец, А. Л. Золкин, Е. А. Свердликова, О. Л. Гиршевич // Экономика и предпринимательство. — 2024. — № 1 (162). — С. 354–359.

86. *Верещагина, Е. А.* Создание безопасных контрактов в технологии блокчейн : монография / Е. А. Верещагина, А. Л. Золкин. — М. : Русайнс, 2022. — 132 с.

87. *Федотов, К. Е.* Конвергенция информационных и транспортных сетей / К. Е. Федотов, А. Л. Золкин // Наука и образование транспорту. — 2020. — № 2. — С. 59–63.

88. *Фролов, А. В.* Администрирование сетей : учебное пособие / А. В. Фролов, Ю. В. Дымченко, А. Л. Золкин. — М. : Русайнс, 2023. — 154 с.

89. Цифровая экономика в России: статистический анализ и прогнозирование : монография / Н. М. Нилова, А. Л. Золкин, С. В. Шамина, И. А. Поскряков. — Краснодар : Новация, 2024. — 175 с.

90. Методы и возможности применения искусственного интеллекта в анализе экономических тенденции / А. О. Кириченко, А. Л. Золкин, Е. А. Свердликова, П. М. Подолько // Прикладные экономические исследования. — 2024. — № 1. — С. 177–184.

91. Packet switching networks simulation / N. N. Vasin, A. O. Kochetova, A. L. Zolkin [et al.] // Third International Conference on Optics, Computer Applications, and Materials Science (CMSD-III 2023). — Washington, 2024. — P. 130650Z.

92. Исследование влияния больших данных на принятие решений в корпоративном секторе / А. О. Кириченко, А. Л. Золкин, А. Б. Урусова, Н. Н. Малова // Журнал прикладных исследований. — 2024. — № 2. — С. 50–57.

93. Применение графовых баз данных в системах поддержки принятия решений / С. Н. Косников, Т. Г. Айгумов, Н. В. Левошич, А. Л. Золкин // Вестник Академии знаний. — 2023. — № 6 (59). — С. 244–248.

94. Безопасность и защита данных в системах поддержки принятия решений / С. Н. Косников, Т. Г. Айгумов, Э. М. Абдулмукминова, А. Л. Золкин // Вестник Академии знаний. — 2023. — № 6 (59). — С. 240–244.

95. Перспективы развития цифровых платформ в глобализированном мире / О. В. Косникова, А. Л. Золкин, В. Е. Ожиганов, В. А. Дорждеева // Индустриальная экономика. — 2024. — № 1. — С. 104–110.

96. *Munister, V. D.* Principles of organizing a hardware of information collection system perceptron for a quantum hashing problem / V. D. Munister, A. L. Zolkin, A. I. Dzhangarov // Proceedings of the III International conference on advances in science, engineering and digital education (ASEDU-III 2022). — Melville, 2024. — P. 50018.

97. Электронные и цифровые деньги: перспективы развития : коллективная монография / Т. Е. Кулумбекова, А. А. Бишенов, К. С. Задорнов [и др.]. — Уфа : ООО «Аэтерна», 2023. — 138 с.

98. Human-centered management in polyergatic information systems. Multi-criteria distribution of functions between operators / E. A. Lavrov, O. E. Siryk, Y. I. Chybiriak [et al.] // IOP Conference Series: Earth and Environmental Science. — 2022. — Т. 1049, № 1.

99. Use of the adaptive neuro-fuzzy system on the example of adjusting the parameters of a neural network in an actuator / V. D. Munister, A. L. Zolkin, V. V. Nagaytsev [et al.] // AIP Conference Proceedings. — AIP Publishing, 2021. — Т. 2402, № 1.

100. *Золкин, А. Л.* Реализация принципов организации и использования средств машинного обучения и искусственного интеллекта в медицине : учебное пособие / А. Л. Золкин, В. Д. Мунистер. — Самара : Мед. ун-т «Реавиз», 2024. — 123 с.

101. Modern models of accounting and analytical support for management decision-making / E. M. Tarasov, S. A. Nadezhkina, A. L. Zolkin [et al.] // E3S Web of Conferences. 2023. — Т. 449. — P. 5002.

102. Применение искусственного интеллекта в оценке рисков заболеваний сердечно-сосудистой системы: новые горизонты и развитие технологий / Т. Н. Чунихина, А. Л. Золкин, Р. А. Вербицкий, В. Е. Ожиганов // Экономика и предпринимательство. — 2024. — № 2 (163). — С. 1137–1143.

103. Оптимизации работы складского комплекса транспортной логистики за счет внедрения механизма сквозных нечетких транзакций / А. Л. Золкин, И. Н. Зайцева, А. С. Тычков, А. А. Артамонов // Научно-технический вестник Поволжья. — 2023. — № 12. — С. 277–283.

104. Mathematical models and decision support system for the efficiency and ergonomic quality of it service management systems / E. Lavrov, V. Borovyk, O. Siryk [et al.] // 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology, PIC S and T 2021s. — 2021. — Vol. 8. — P. 506–510.

105. *Золкин, А. Л.* Цифровой мониторинг агроэкосистем на основе космических и беспилотных технологий как основа органического земледелия / А. Л. Золкин, Е. В. Матвиенко. — М. : Русайнс, 2023. — 66 с.

106. *Золкин, А. Л.* Обоснование применения современных летательных аппаратов и средств в технологических операциях сельского хозяйства : монография / А. Л. Золкин, Е. В. Матвиенко, Ю. В. Осоргин. — М. : Русайнс, 2023. — 122 с.

107. Технология распределенных реестров «блокчейн» в АПК / Э. Ф. Амирова, Е. А. Колобанова, А. Л. Золкин, А. О. Шилин // Научно-технический вестник Поволжья. — 2023. — № 7. — С. 109–115.

108. Перспективы использования беспилотных технологий в сельском хозяйстве / Э. Ф. Амирова, Р. И. Вагапов, А. Л. Золкин, Н. Н. Малова // Научно-технический вестник Поволжья. — 2023. — № 8. — С. 41–48.

109. Разработка автоматизированной системы управления перевозками грузов с внедрением аппарата нечеткого управления / А. Л. Золкин, Л. В. Куныгина, Е. А. Попова, И. В. Журавлева // Научно-технический вестник Поволжья. — 2023. — № 7. — С. 142–148.

110. *Золкин, А. Л.* Создание программного обеспечения для нательных компьютерных сетей и носимых систем : учебное пособие / А. Л. Золкин. — Самара : Мед. ун-т «РЕАВИЗ», 2024. — 142 с.

111. *Верещагина, Е. А.* Методы и алгоритмы организации процессов мониторинга информационных систем : монография / Е. А. Верещагина, А. Л. Золкин. — М. : Русайнс, 2024. — 152 с.

112. *Золкин, А. Л.* Разработка компьютерных сетей с внедрением микросервисной архитектуры для телемедицинского обеспечения : учебное пособие / А. Л. Золкин, В. Д. Мунистер. — Самара : Мед. ун-т «РЕАВИЗ», 2024. — 168 с.

113. *Изаак, Р. В.* Автоматизация и ее интеграция в сложные технологические процессы / Р. В. Изаак, А. Л. Золкин // Проблемы и перспективы внедрения инновационных телекоммуникационных технологий : материалы X Международной научно-практической конференции. — Оренбург, 2024. — С. 272–282.

114. *Изаак, Р. В.* Анализ методов моделирования сложных систем, применяемых в производственных процессах / Р. В. Изаак, А. Л. Золкин // Проблемы и перспективы внедрения инновационных телекоммуникационных технологий : материалы X Международной научно-практической конференции. — Оренбург, 2024. — С. 260–272.

115. Внедрение улучшенного мультисервисного обслуживания для узла доступа с внедрением нейро-нечеткого программного модуля / А. Л. Золкин, Р. А. Вербицкий, А. С. Тычков, Г. Ю. Азаренко // Научно-технический вестник Поволжья. — 2024. — № 2. — С. 69–73.

116. Реализация программного контроллера управления процессом обработки данных системы рекультивации почвы агрокомплекса / А. Л. Золкин, Е. В. Матвиенко, С. В. Шамина, Ю. Н. Коваль // Научно-технический вестник Поволжья. — 2024. — № 2. — С. 74–79.

117. Разработка инструментальных средств создания программного обеспечения для портативных биоинформационных устройств с

RFID-чипами / А. Л. Золкин, А. В. Юмашев, Н. Ю. Логунова, К. С. Задорнов // Научно-технический вестник Поволжья. — 2024. — № 2. — С. 85–90.

118. Особенности конфигурирования паролей маршрутизаторов в сетях передачи данных на железнодорожном транспорте / А. Е. Горбунов, А. Л. Золкин, С. А. Надежкина, В. А. Надежкин // Научно-технический вестник Поволжья. — 2024. — № 3. — С. 79–83.

119. Создание мультипараметрической системы обслуживания заявок в многофункциональном центре обработки персональных данных / А. Л. Золкин, В. Д. Мунистер, О. В. Сараджева, Ф. Ф. Хабибуллин // Научно-технический вестник Поволжья. — 2024. — № 3. — С. 88–92.

120. Разработка инструментальных средств создания биоинформаццонного программного обеспечения для нательных носимых устройств / А. Л. Золкин, А. В. Юмашев, А. Н. Корнетов, Ф. Р. Ахмадуллин // Научно-технический вестник Поволжья. — 2024. — № 3. — С. 93–97.

121. *Золкин, А. Л.* Информатика : учебное пособие / А. Л. Золкин. — Самара : Мед. ун-т «РЕАВИЗ», 2023. — 104 с.

122. *Тормозов, В. С.* Основы работы в операционной системе Linux : учебное пособие / В. С. Тормозов, А. Л. Золкин. — Самара : Мед. ун-т «РЕАВИЗ», 2023. — 66 с.

123. *Золкин, А. Л.* Основы алгоритмизации, мировые информационные ресурсы, медико-биологическая статистика : учебное пособие для студентов медицинских вузов / А. Л. Золкин. — Самара : Мед. ун-т «РЕАВИЗ», 2022. — Ч. 1. — 161 с.

Александр Леонидович ЗОЛКИН

ПРОГРАММИРОВАНИЕ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ

УЧЕБНИК

Зав. редакцией литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *Е. О. Сапарова*
Подготовка макета *Д. А. Вакулова*
Корректор *Т. А. Кошелева*
Выпускающий *В. А. Иутин*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028 от 14.04.2016 г.,
выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр-кт Ю. Гагарина, д. 1, лит. А.
Тел./факс: (812) 336-25-09, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 22.01.25.
Бумага офсетная. Гарнитура Школьная. Формат 84×108 $^{1}/_{32}$.
Печать офсетная/цифровая. Усл. п. л. 7,77. Тираж 30 экз.

Заказ № 065-25.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский пр-кт, д. 42, к. 5.