Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина
Уральский энергетический институт

А. М. ЗЮЗЕВ
М. В. МУДРОВ
А. В. КОСТЫЛЕВ

ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ КОНТРОЛЛЕРЫ В СИСТЕМАХ МАЛОЙ АВТОМАТИЗАЦИИ И АВТОМАТИЗАЦИИ ЗДАНИЙ И СООРУЖЕНИЙ

Учебное пособие

Министерство науки и высшего образования
Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

А. М. Зюзев, М. В. Мудров, А. В. Костылев

ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ КОНТРОЛЛЕРЫ В СИСТЕМАХ МАЛОЙ АВТОМАТИЗАЦИИ И АВТОМАТИЗАЦИИ ЗДАНИЙ И СООРУЖЕНИЙ

Учебное пособие

Под общей редакцией доктора технических наук А. М. Зюзева

Рекомендовано методическим советом
Уральского федерального университета для студентов вуза,
обучающихся по направлению подготовки
13.03.02 — Электроэнергетика и электротехника

Екатеринбург
Издательство Уральского университета
2023

УДК 681.5:69.03(075.8)
ББК 32.965я73+38.71я73
З-98

Рецензенты:
кафедра электрификации горных предприятий Урал. гос. горного ун-та (завкаф. канд. техн. наук, доц. *М. Е. Садовников*);
директор ООО «Тяжпромэлектропривод» канд. техн. наук, доц. *В. И. Зеленцов*

Зюзев, Анатолий Михайлович.

З-98 Программируемые логические контроллеры в системах малой автоматизации и автоматизации зданий и сооружений : учебное пособие / А. М. Зюзев, М. В. Мудров, А. В. Костылев ; под общ. ред. д-ра техн. наук А. М. Зюзева ; М-во науки и высшего образования РФ. — Екатеринбург : Изд-во Урал. ун-та, 2023. — 118, [2] с. : ил. — ISBN 978-5-7996-3788-0. — Текст : непосредственный.

ISBN 978-5-7996-3788-0

В пособии рассматриваются особенности применения программируемых логических контроллеров (ПЛК) как основных управляющих устройств системы автоматизации зданий и систем малой промышленной автоматизации; последовательно излагается история появления и развития зарубежных и отечественных ПЛК, базовые основы логического управления, некоторые сетевые технологии, применяемые в малой автоматизации. Приводится ряд примеров применения ПЛК в типовых задачах автоматики.

Предназначено для бакалавров, изучающих дисциплину «Автоматизация зданий и сооружений» по направлению подготовки 13.03.02.

Табл. 19.

УДК 681.5:69.03(075.8)
ББК 32.965я73+38.71я73

ISBN 978-5-7996-3788-0

© Уральский федеральный университет, 2023

Оглавление

Введение **5**

Глава 1. Основные принципы построения программируемых логических контроллеров **8**
1.1. Общая информация 8
1.1.1. Предпосылки создания ПЛК 8
1.1.2. История развития ПЛК 10
1.2. Структура ПЛК 14
1.3. Функциональные возможности 15
1.4. Языки и системы программирования ПЛК 16
1.4.1. Языки программирования ПЛК 18
1.4.2. Среды программирования распространенных ПЛК 22
1.5. Области применения ПЛК 24
Конрольные темы для повторения 25

Глава 2. Основы логического управления **26**
2.1. Описание дискретных входов 26
2.2. Описание дискретных выходов 28
2.2.1. Релейные выходы 28
2.2.2. Транзисторные выходы 29
2.3. Примеры программирования дискретных входов-выходов 32
2.3.1. Язык C 32
2.3.2. Язык LD 36
2.3.3. Язык FBD 37
Конрольные темы для повторения 37

Глава 3. Применение ПЛК в распределенных системах автоматизации . **38**
3.1. Сетевая модель OSI 38
3.2. Применяемые в промышленности цифровые сети 40
3.2.1. ModBus 41
3.2.2. Controller Area Network 67

3.3. Примеры программирования 77
3.3.1. Программирование ModBus на языке C 77
3.3.2. Программирование CAN на языке C 86
Конрольные темы для повторения 89

Глава 4. Пример применения ПЛК **90**
4.1. Описание учебного стенда 90
4.2. Оборудование и подготовка учебного стенда к работе 91
4.2.1. Оборудование 91
4.2.2. Цифровые интерфейсы 97
4.2.3. Подготовка к работе 99
4.3. Настройки в системе Windows 99
4.4. Обзор переменных контроллера для удаленного управления . . . 101
4.4.1. Установка системы SCADA 102
4.4.2. Настройки в системе SCADA 105
4.4.3. Проверка работы в системе SCADA 110
Конрольные темы для повторения 115
Список библиографических ссылок 116

Введение

Говоря о применении программируемых логических контроллеров во всех сферах промышленной автоматизации, следует обратиться к 1970-м гг. В истории науки и техники этот период называют третьей промышленной революцией, в результате которой возникает такое понятие, как «Индустрия 3.0». Очень часто именно данный период называют периодом первой цифровой революции, когда вычислительные устройства, уже широко применяемые в научных и технических расчетах, начинают активно использовать и в задачах управления. Именно тогда появляется термин «контроллер» – микропроцессорное устройство, предназначенное для решения задач управления процессом (от англ. *to control* – управлять). Учитывая характер решаемых задач, термин приобретает современное звучание – программируемый логический контроллер, или ПЛК. Логическое или же дискретное управление предполагает обработку дискретных сигналов типа включения катушки исполнительного устройства (реле, контактора или электромагнита), опроса состояния дискретных датчиков, которые могут формировать только два уровня сигнала, соответствующих двум состояниям объекта, (например, датчик перегрузки в лифте), опроса многочисленных контактов и кнопок, также имеющих два состояния (включено – выключено). Для синтеза алгоритмов управления подобных систем с дискретными элементами широко применялся (еще до появления ПЛК) математический аппарат алгебры логики, что и дало им название «алгоритмы логического управления». Соответственно контроллеры унаследовали это наименование.

С 1970-х гг. промышленная автоматизация совершила колоссальный скачок в развитии. Пожалуй, главной особенностью современной автоматизации стала сложная многоуровневая иерархия систем управления. Кроме дискретных датчиков (иногда используют англицизм «sensor» – «сенсор» и исполнительных устройств (англицизм «actuator» – «актуатор»), появляются другие классы устройств, систем и подсистем. То, что составляло ядро промышленной автоматики, сегодня относят к системам малой автоматизации. Данные системы характеризуются преимущественно дискретными элементами «актуатор – сенсор» и относительно простыми алгоритмами управления. Отметим, что по количеству устройств здесь как раз нет ограничений: это могут быть

достаточно обширные и разветвленные системы, например система контроля парковочных мест в крупных торговых центрах. Если место освобождается, о чем известил датчик присутствия, над ним включается зеленая сигнальная лампочка, а счетчик свободных мест меняет своё значение.

Приведенный пример не случаен. Развитие автоматизации уже в кон. 1980–90-х гг. приводит к тому, что она начинает охватывать не только классическое производство. Автоматизируются сопутствующие процессы, например, складские системы, системы промышленного освещения и т. п. Начинается процесс автоматизации зданий. Фраза известнейшего архитектора первой половины XX в. Ле Корбюзье «Дом – это машина для жилья» приобретает прямое значение. Интересно отметить, что процесс автоматизации зданий шел с двух сторон и реализуется до сих пор в двух существенно различных парадигмах. С одной стороны, классическая технология автоматизации, привнесенная из промышленности, с другой – семейство технологий из IT-сферы, например технологии «умного дома» и «интернета вещей». Причем более доступные обычному человеку и дешевые решения IT-парадигмы в домашней бытовой автоматизации распространены больше. Между тем, отличия этих подходов, при внимательном рассмотрении, скорее не в базовых принципах автоматики, а в их реализации. Это также логическое управление, основанное на взаимодействии датчиков и исполнительных устройств под управлением контроллера, разве что последний в IT-решениях может быть виртуальным и реализован программно на удаленном сервере.

Важно отметить, что современные системы автоматизации всех уровней являются, как правило, распределенными. Это предполагает наличие особой сетевой среды удаленного обмена информацией между всеми типами устройств. В малой автоматизации основными требованиями к такой среде являются простота реализации и, разумеется, надежность в широком понимании этого термина. Конкуренция между отдельными производителями оборудования, альянсами производителей привела к появлению очень большого числа вариантов реализации такой сетевой среды. Это могут быть технологии на основе компьютерных сетей (Ethernet и Wi-Fi), специальные проводные и беспроводные интерфейсы (Z-wave, ZigBee и т. п.), а кроме того, решения, унаследованные из промышленной автоматизации.

Одним из первых решений в области промышленных сетей стала сеть Modbus, разработанная еще в 1970-е гг. практически вместе с первым поколением ПЛК. Эта технология отличается прежде всего простотой и понятностью реализации, что выгодно отличает ее от последующих решений. Построение распределенной системы малой автоматизации на основе сети Modbus выполняется без избыточных усилий разработчика как в проектном, так и в экономическом аспектах. Начать знакомство с сетевыми средами, безусловно, удобнее с этого интерфейса. Тем не менее данное решение имеет и очевидные недостатки. В структурном плане главным ограничением для применения сети в системах с большим количеством объектов автоматизации является наличие только одного управляющего устройства, что отрицательно сказывается на общей живучести и производительности системы. Более современные технологии уже не имеют этого недостатка, например сеть на основе интерфейса CAN, послужившая основой для целого ряда решений, в т. ч. в автоматизации зданий и бортовой автоматизации транспортных систем.

Возвращаясь к проблеме столкновения двух парадигм управления в малой автоматизации и автоматизации зданий, необходимо упомянуть и различие в средах разработки. Так, решения, выполненные в парадигме классической автоматизации, опираются на стандарт для языков программирования МЭК 61131-3. Языки этого стандарта ориентированы на инженеров в области автоматизации, они предлагают прежде всего визуальные средства программирования, близкие к схемам автоматики. Это очевидное удобство является одновременно существенным ограничением для реализации сложных нестандартных алгоритмов. Впрочем, для решений малой автоматизации они вполне эффективны.

Подход IT-специалистов предполагает использование классических языков программирования, традиционно близких к аппаратным средствам – «железу». Это прежде всего семейство языков «C». Встречаются и специализированные языки, разработанные специально для некоторых контроллеров. Применение классического языка дает гораздо большую свободу разработчику, но при этом требует и более высокой подготовки специалиста в программировании.

Таким образом, современная малая автоматизация представляет собой вполне четко очерченную и динамично развивающуюся область, освоение которой важно для специалистов в области промышленной автоматизации.

Глава 1. Основные принципы построения программируемых логических контроллеров

1.1. Общая информация

Контроллер (от *англ.* Control – управление) – это техническое устройство в составе автоматизированных комплексов, которое способно управлять физическими процессами по заданному алгоритму с применением информации, получаемой от датчиков, и с формированием сигналов управления, выводимых на исполнительные устройства. Любое устройство, способное работать автоматически, имеет в своем составе управляющий модуль, определяющий логику работы устройства.

Программируемый логический контроллер (ПЛК) представляет собой микропроцессорное устройство, применяемое при сборе данных, их преобразовании и обработке, способное хранить информацию и выдавать команды управления. ПЛК имеет конечное количество входов и выходов, возможность подключения различных датчиков и исполнительных устройств и предназначен для работы в режиме реального времени.

ПЛК, используемый для автоматизации технологических процессов, представляет собой блок, который выполняет операции над дискретными или непрерывными сигналами в различных условиях. Первоначально он применялся для замены релейно-контактной аппаратуры в промышленности. В настоящее время ПЛК используются практически во всех отраслях промышленности.

1.1.1. Предпосылки создания ПЛК

История ПЛК начинается в 1960-х гг., когда системы управления строились на основе релейно-контакторных схем управления, которые состояли из модулей, содержащих множество реле, клеммных колодок и массу проводов.

Проблем, связанных с подобными системами, было множество, среди них можно отметить:

– отсутствие гибкости при масштабировании работающего комплекса, а также много времени, требующегося для корректировки алгоритма автоматизации технологического процесса, когда необходимы изменения;

– проблемы, обусловленные низкой надёжностью электромагнитных реле, контактных соединений проводов, быстро стареющей маркировкой на клеммных колодках и т. д.

С указанными проблемами постоянно сталкивались техники и инженеры, тогда же родилась поговорка «Пять часов, чтобы найти, и пять минут, чтобы исправить».

В 1968 г. инженеры подразделения Гидраматик корпорации «Дженерал Моторс» представили на конференции «Вестингауз» доклад, в котором изложили список проблем, связанных с надежностью и ведением документации для машин завода, и представили критерии проектирования, разработанные инженерами GM для «стандартного контроллера».

Согласно разработанным критериям, первая модель контроллера должна удовлетворять следующим требованиям:

– исключение дорогостоящего брака реле при смене моделей;
– замена ненадежных электромеханических реле;
– распространение статических цепей (на основе электронных компонентов) вместо электромеханических реле на 90 % машин на заводе;
– сокращение времени простоя оборудования, связанного с проблемами управления, – необходима простота обслуживания и программирования в соответствии с уже принятыми схемами релейной логики;
– возможность расширения в будущем – управляющая система должна быть модульной, чтобы можно было легко заменять компоненты и расширять функции;
– возможность работы системы управления в промышленных условиях загрязнения, влаги и вибраций.

Перечисленные требования, совместно с запросом на создание прототипа, были переданы четырем компаниям-сборщикам систем управления: «Аллен-Брэдли» через компанию «Информэйшн Инструментс» из Мичигана, «Диджитал Эквипмент Корпорэйшн», «Сенчури Детройт», «Бэдфорд Ассошиэтс».

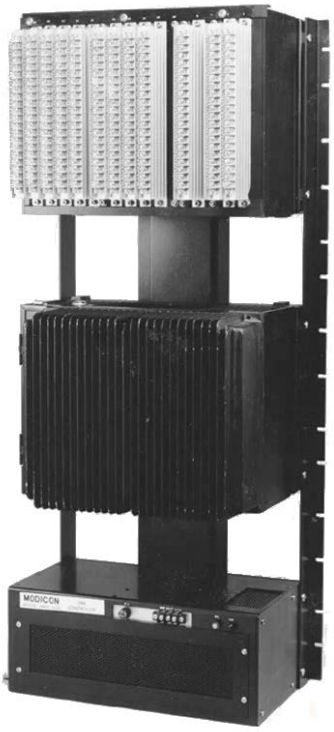
1.1.2. История развития ПЛК

Получив запрос от комании «Дженерал Моторс», команда «Диджитал Эквипмент» представила в «Дженерал Моторс» мини-компьютер, который в итоге был отвергнут по множеству причин, одной из которых было серьезное ограничение памяти.

«Аллен-Брэдли», уже хорошо известная своими реостатами, реле и средствами управления двигателями, отреагировала на запрос, рискуя конкурировать с одним из своих самых успешных основных направлений деятельности – производством электромеханических реле. Желая выполнить все требования, «Аллен-Брэдли» перешла от прототипа контроллера к его фактическому производству за 5 мес. Первая модель, PDQ-II (программный квантователь данных) была сочтена слишком большой и слишком сложной для программирования. Вторая модель, PMC (программируемый матричный контроллер) была меньше и проще в программировании, но всё же не могла полностью удовлетворить потребности клиентов в задачах управления механообрабатывающими станками.

К моменту подачи предложения от GM, инженеры из «Бэдфорд Ассошиэтс» уже работали над конструкцией устройства, характеристики которого включали модульную и прочную конструкцию, отсутствие прерываний на обработку данных, а также прямое обращение к памяти. Команда «Бэдфорд Ассошиэтс» назвала этот агрегат 084, т. к. это был 84-й проект компании. Найдя финансовую поддержку, команда решила создать новую компанию под названием «Модикон» (Modicon – MOdular DIgital CONtroller), которая тесно сотрудничала с «Бэдфорд Ассошиэтс» для создания контроллера. Команда «Модикон» заканчивала проектирование и сборку 084, который теперь они называли программируемым контроллером.

В итоге в 1969 г. победившее предложение поступило от компаний «Бэдфорд Ассошиэтс» и «Модикон», когда они продемонстрировали на выставке GM полупроводниковый последовательный логический вычислитель Modicon 084, состоявший из трех отдельных компонентов, включая плату процессора, память и плату логического решающего устройства, которые выполняли основные алгоритмы, связанные с релейной логикой.



Первый ПЛК MoDiCon 084

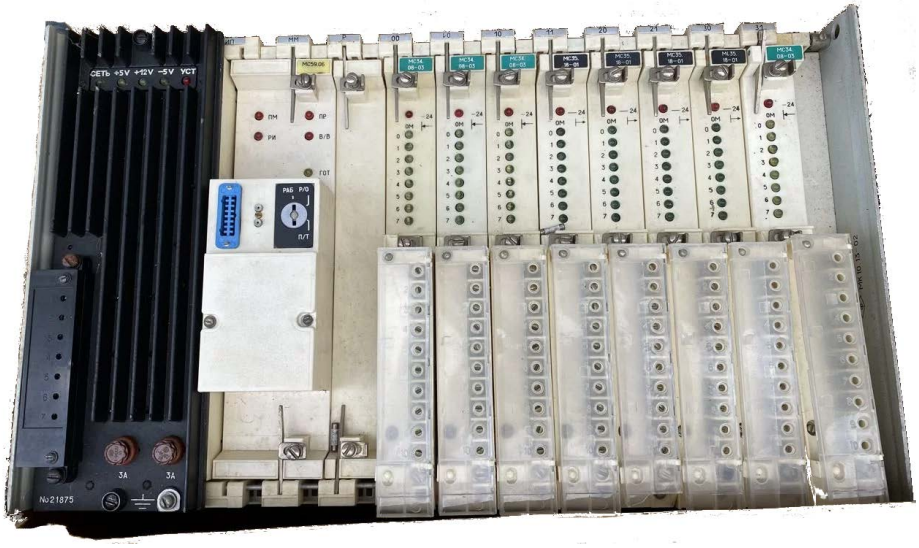
Modicon 084 был прочным, полностью закрытым и снабженным кондуктивным охлаждением, что делало его надежным. Создатели объясняли: «Вентиляторы не использовались, а наружный воздух не попадал в систему из-за опасений загрязнения и коррозии. Мысленно мы представляли, что программируемый контроллер находится под грузовым автомобилем, на открытом воздухе и ездит по Техасу, по Аляске. В этом случае мы хотели, чтобы он остался целым. Другим требованием было то, чтобы он стоял на столбе, помогая управлять коммунальным предприятием или микроволновой станцией, которая не имела климат-контроля и вообще не обслуживалась».

К 1971 г. инженеры из «Аллен-Брэдли» приступили к разработке новой модели, включавшей улучшения, основанные на потребностях клиентов, которые не могли быть реализованы в PMC. Эта новая модель была известна как Bulletin 1774 PLC. Компания «Аллен-Брэдли» назвала это новое устройство

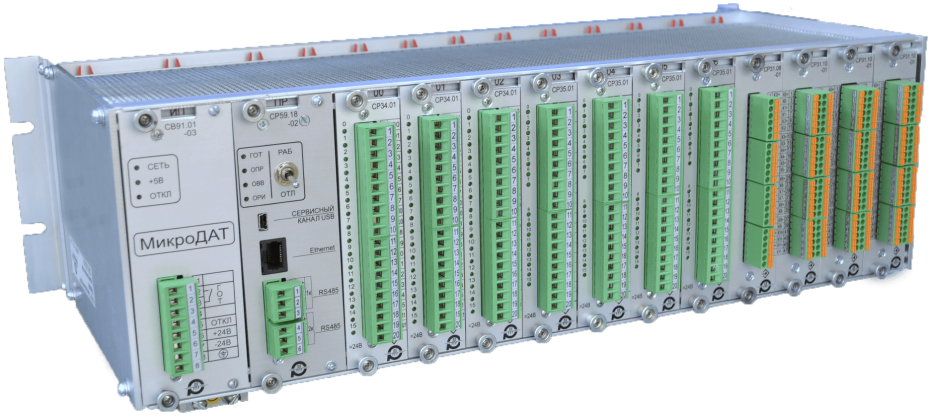
программируемым логическим контроллером (ПЛК) вместо принятого тогда термина «программируемый контроллер». ПЛК стала промышленным стандартом, особенно когда ПК стали ассоциироваться с персональными компьютерами. В 1985 г. «Роквелл Автомешн» приобрела «Аллен-Брэдли». Название выпускаемой новинки до сих пор ассоциируется с «Аллен-Брэдли».

В это время «Модикон» уже набирался опыта благодаря Modicon 084, и в 1973 г. появился Modicon 184. Новая модель не только отвечала потребностям рынка и клиентов, но и способствовала взлету компании «Модикон», сделав ее первым лидером на рынке. В 1977 г. компания «Модикон» была продана «Гоулд Электроникс», а позже, в 1997 г., была продана компании «Шнайдер Электрик», которая до сих пор владеет брендом и использует имя Modicon.

В СССР в 1984 г. было освоено массовое производство первых отечественных программируемых логических контроллеров «МикроДАТ» [1]. Десятки тысяч контроллеров «МикроДАТ», сошедших с конвейеров заводов еще Советского Союза, и сегодня успешно работают на предприятиях России и стран СНГ. Производство этих контроллеров находилось в г. Александрии и г. Харькове, в 2007 г. было перенесено в г. Белгород, где и находится сейчас.

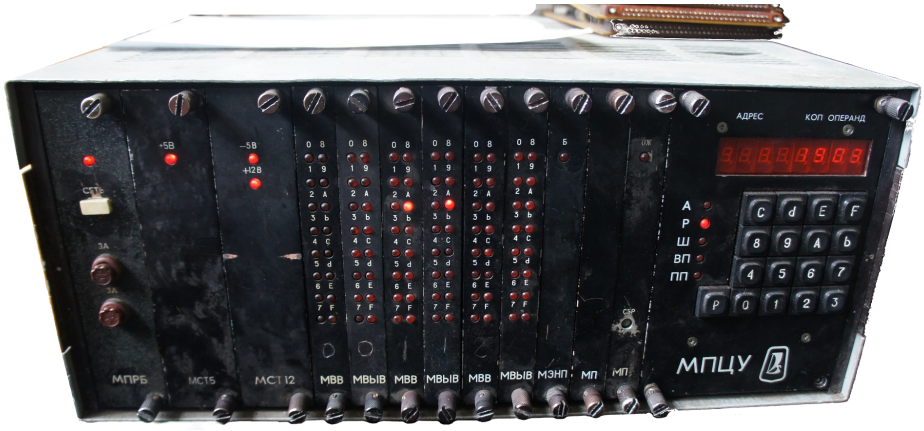


Первый ПЛК «МикроДАТ»



Современный ПЛК «МикроДАТ»

Одновременно с «МикроДАТ» в г. Тольятти, на производстве Волжского автомобильного завода (ВАЗ), были разработаны и внедрены свои линейки ПЛК – микропроцессорного программируемого циклового устройства, сокращенно МПЦУ-2-48, которые успешно справлялись с задачами автоматизации сложных технических комплексов производства ВАЗа.



ПЛК МПЦУ-2-48

Сегодня как зарубежные, так и отечественные компании ведут работу в направлении развития ПЛК, дополняя существующие модели различными интерфейсами, усиливая надежность, улучшая программное обеспечение, при помощи которого пользователь работает с контроллерами.

1.2. Структура ПЛК

Структура ПЛК может варьироваться в зависимости от производителя и модели, но обычно она включает следующие основные компоненты (рис. 1.1):

– центральный процессор (ЦПУ) – это основной компонент ПЛК, который выполняет программы управления и координирует работу всех других компонентов;
– память – используется для хранения программ управления и параметров;
– входы-выходы – входы и выходы, которые позволяют им получать данные от датчиков и других источников входных сигналов, а также управлять механизмами и другими устройствами через выходы;
– интерфейс – интерфейс, который позволяет программисту создавать программы управления и конфигурировать ПЛК для различных задач;
– блок питания – ПЛК требуется электропитание для работы, поэтому они обычно имеют встроенный блок питания или работают от внешнего источника питания;
– коммуникационные порты – некоторые модели ПЛК могут иметь различные коммуникационные порты, которые позволяют ПЛК обмениваться данными с другими устройствами или контролировать их;
– специальные модули – некоторые ПЛК имеют специальные модули, которые могут использоваться для управления определенными задачами, такими как управление движением и др.

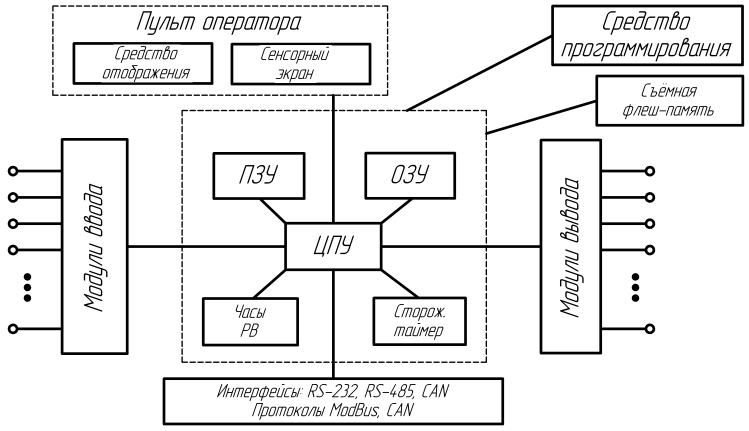


Рис. 1.1. Типовая структура ПЛК

Структура ПЛК может быть разнообразной, в зависимости от модели и производителя, но все они обеспечивают гибкое и надежное управление производственными процессами. Программист, работая над программным алгоритмом и сохраняя его в памяти ПЛК, определяет, каким образом ПЛК будет реагировать на изменение состояния входных сигналов. После загрузки программного кода операционная система ПЛК обеспечивает циклическое выполнение программы.

Помимо классической периферии (дискретной, аналоговой), ПЛК оснащается интерфейсом для связи с программатором, таким же или другим интерфейсом для связи с оператором (при необходимости) и сетевыми интерфейсами.

1.3. Функциональные возможности

Ниже перечислены основные функциональные возможности ПЛК:

- логическое управление, позволяющее выполнять различные логические операции, такие как «И», «ИЛИ», «НЕ» и др.;
- математические операции, такие как сложение, вычитание, умножение и деление;
- аналоговый контроль – ПЛК могут оперировать непрерывными сигналами, такими как напряжение и ток;
- дискретный контроль – ПЛК могут контролировать и управлять дискретными сигналами, такими как сигналы от датчиков и кнопок;
- позиционное управление – ПЛК могут контролировать и управлять позиционными системами, такими как шаговые двигатели или сервоприводы;
- циклическое управление – ПЛК позволяют управлять циклическими процессами, такими как переключение режимов работы оборудования в зависимости от времени;
- программирование – ПЛК обычно программируются на языке логических блоков (Ladder Logic), но могут также быть запрограммированы на других языках, таких как C++ или Java;
- связь с другими устройствами – ПЛК могут работать в сетях, обмениваясь данными с другими устройствами, такими как ПК, счетчики, датчики;
- мониторинг процессов, выполняемых на оборудовании, и вывод информации на дисплей или отправка ее на другие устройства.

Программируемые логические контроллеры широко используются в сфере промышленной автоматизации всевозможных технологических процессов на крупных и небольших предприятиях. Их распространённость объясняется легко. Использование ПЛК весьма упрощает создание и использование как комплексных автоматических систем, так и отдельных приборов, в т. ч. для домашнего предназначения. Они позволяют уменьшить период разработки, упрощают процесс монтажа и отладки за счет стандартизации отдельных аппаратных и программных компонентов, а также гарантируют повышенную надежность в процессе эксплуатации и, при необходимости, несложный ремонт и модернизацию.

Логические контроллеры в большом количестве встречаются на автоматических и сборочных линиях. Их можно встретить на стендах для испытания двигателей, на прессах, токарных автоматах, сварочных аппаратах, устройствах для резки металла. Они также используются при автоматизации технологических процессов быта. Нынешний ПЛК стал нужным универсальным рабочим инвентарем в системах автоматизации производственных процессов, а также средством управления отдельными приборами различного предназначения. Это определенный образ программируемых автоматов, отличающийся повышенной надежностью, просто встраиваемый и модернизируемый, способный долговременно трудиться буквально без обслуживания.

1.4. Языки и системы программирования ПЛК

В 1993 г. Международной электротехнической комиссией (МЭК) был опубликован стандарт МЭК 61131-3 (IEC 61131-3) [2]. Этот международный стандарт входит в группу стандартов МЭК 1131, которые рассматривают различные варианты использования ПЛК. Назначение МЭК 61131-3 – стандартизация существующих языков программирования ПЛК, а вернее, базовая платформа для такой работы в национальных комитетах стандартизации.

Стандарт МЭК 61131-3 оказался настолько актуален, что функции поддержки и внедрения стандарта на рынке взяла на себя независимая организация ПЛЦОпен, состоящая из производителей и пользователей программного обеспечения (ПО), ориентированного на МЭК 61131-3. В результате деятель-

ности ПЛЦОпен на рынке ПО появилась серия сертифицированных средств программирования ПЛК, которые достаточно широко и небезуспешно внедряются в промышленности. Список инструментальных программных систем, реализующих стандарт МЭК 61131-3, превышает два десятка.

Примеры инструментальных систем, реализующих МЭК 61131-3:

МЭК-система	**Фирма**
CoDeSys	Smart Software Solution (www.3S-software.com), Германия
ACCON-ProSys	Deltalogic (www.deltalogic.de), Германия
OpenPCS	Infoteam Software (www.infoteam.de), Германия
PUMA	KEBA (www.keba.com), Австрия
SUCOsoftS340	Klokner-Moeller (www.moeller.net), Германия
NAIS CONTROL	Matsushita Electric Works AG, Германия
PDS7	Philips Industrial Electronics Services, Нидерланды
SELECONTROL	Selectron Lyss, Швейцария
4Control	Softing (www.softing.com), Германия
ISaGRAF	CJ International (www.isagraf.com), Франция

Стандарт МЭК 61131-3 определяет языки для программируемых контроллеров таким образом, что части прикладной программы могут быть записаны на любом языке и скомпонованы в единый исполняемый код. При разработке стандарта было найдено так много вариаций языков для программируемых контроллеров, что было невозможно выбрать одну из существующих вариаций в качестве общего языка. Новый стандарт включает структурное программирование, абстрактные типы данных, выделение данных и процедур в блок (инкапсуляцию) в сочетании с сохранением тесной связи с классическими языками для программируемых контроллеров.

Необходимо отметить, что выбор используемого языка довольно часто определяется личными предпочтениями пользователя и мало связан с технологическим процессом, который необходимо автоматизировать. Представленные в стандарте языки в большинстве случаев являются взаимозаменяемыми. Это значит, что при разном уровне подготовленности в области программирования пользователи могут создавать программы равной функциональности.

1.4.1. Языки программирования ПЛК

Языки программирования стандарта МЭК 61131-3 включают в себя 3 визуальных языка (FBD, SFC, LD), ориентированных на инженеров и бизнес-аналитиков, и 2 текстовых (ST, IL), ориентированных на программистов, (рис. 1.2).

Во многих инструментальных системах «61131» присутствует возможность смешивать программы, функции, процедуры, которые написаны на разных языках, а также вставлять кодовые последовательности из одного языка в коды, написанные на другом языке. Любое функциональное расширение возможно за счет поддержки СИ-интерфейса, который присутствует во многих средах. В итоге МЭК 61131-3 (IEC 61131-3) содержит богатый набор стандартных функций:

– функции для работы с битовыми переменными,
– числовые функции,
– функции преобразования типов,
– функции сравнения,
– функции, определяемые производителем и пользователем.

Также в стандарте МЭК 61131-3 содержатся функциональные блоки:

– блоки синхронизации состояний,
– блоки выделения переднего и заднего фронта дискретного сигнала,
– таймеры и счетчики,
– функциональные блоки, определяемые пользователем и производителем.

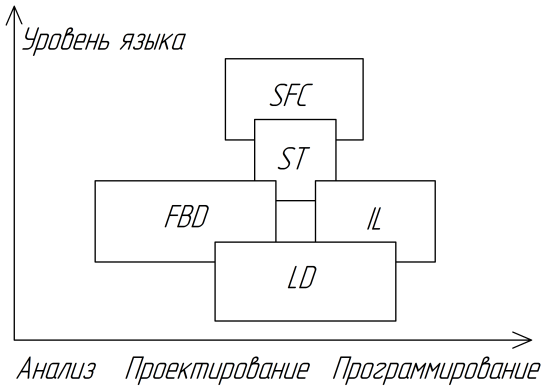


Рис. 1.2. Место языков на этапах разработки программы для ПЛК

При помощи перечисленных блоков и функций возможно выполнение операций над большим количеством стандартных типов данных и переменных, таких как булевы, целые, беззнаковые целые, вещественные, строки символов, массивы и др.

Далее представлены примеры программы управления насосной станцией. В примере I1 и I2 – датчики нижнего и верхнего уровня воды в резервуаре, а Q1 – работа насоса. При подаче напряжения на схему, насос включается и вода начинает заполнять резервуар. По достижении водой датчика нижнего уровня, отключается инверсный контакт I1, но насос остается включенным по цепи контакта Q1. В тот момент, когда вода достигнет датчика верхнего уровня, насос отключается сигналом датчика верхнего уровня I2. Когда вода израсходована из резервуара, т. е. уровень воды находится ниже датчика нижнего уровня, насос снова включается, и цикл повторяется, поддерживая уровень воды между датчиками нижнего и верхнего уровней.

Язык релейно-контактных схем (LD)

Язык релейных диаграмм или релейной логики (Ladder Diagrams) применяется для описания логических выражений различного уровня сложности и использует в качестве базовых элементов программирования графические элементы – «контакты» (contacts) и «катушки» (coils), связанные с входными и выходными каналами соответственно.

Присутствие в стандарте языка LD определяется, скорее всего, данью традициям, т. к. для релейной техники было разработано огромное количество оборудования и алгоритмов (рис. 1.3). Сегодня, имея типовой набор цифрового ввода-вывода, можно создавать управляющие системы на отлаженной годами алгоритмической базе.

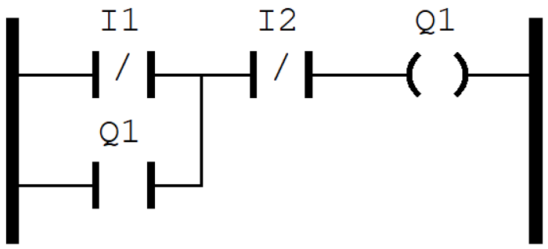


Рис. 1.3. Пример кода программы на языке релейно-контактных схем LD

Язык функциональных блоков (FBD)

Язык функциональных блоков (Function Block Diagrams) позволяет создать программную единицу практически любой сложности на основе стандартных блоков (арифметические, логические блоки, блоки, описывающие законы управления, и т. д.). На рис. 1.4 показан пример типичных элементов. Данное средство использует технологию инкапсуляции алгоритмов обработки данных и законов регулирования. В итоге программирование на языке FBD сводится к «склеиванию» готовых компонентов. В результате получается максимально наглядная и хорошо контролируемая программная единица.

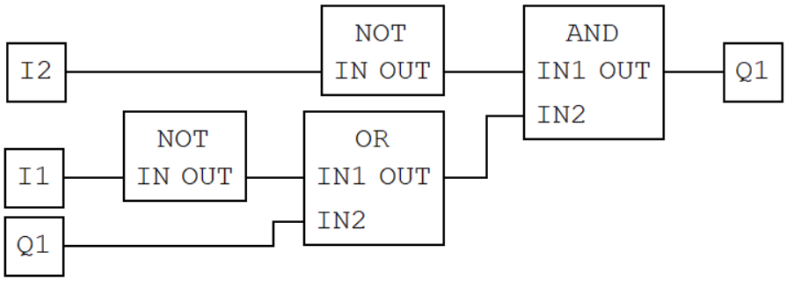


Рис. 1.4. Пример кода программы на языке функциональных блоков FBD

Язык структурированного текста (ST)

Язык структурированного текста (Structured Text) относится к классу текстовых языков высокого уровня. Этот язык уходит корнями в такие известные языки программирования, как Ada, Pascal и C. На его основе можно создавать гибкие процедуры обработки данных. Язык структурированного текста является основным для программирования последовательных 73 шагов и транзакций языка SFC. Кроме этого, он имеет выходы во все остальные языки, что делает его универсальным для разных категорий пользователей. Пример кода программы на языке структурированного текста представлен ниже.

```
1    Q1 := NOT(I2) AND (NOT(I1) OR Q1);
```

Язык последовательных функциональных схем (SFC)

Язык последовательных функциональных схем (Sequential Function Charts или Grafcet) позволяет формулировать логику программы на основе чередующихся процедурных шагов и транзакций (условных переходов) (рис. 1.5), а также описывать последовательно-параллельные задачи в понятной и наглядной форме.

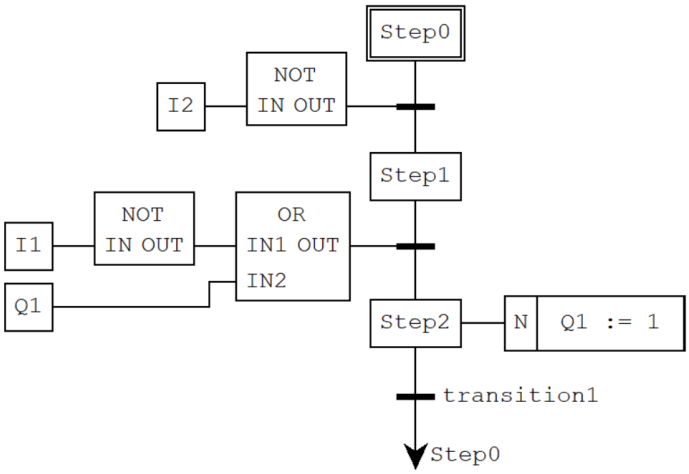


Рис. 1.5. Пример кода программы на языке последовательных функциональных схем SFC

Строго говоря, SFC не является языком программирования. Это средство проектирования прикладного программного обеспечения, которое является комплексом большого числа программных единиц: программ, функциональных блоков, функций. Обеспечение параллельности выполнения программ, установление и контроль состояния порожденных процессов, обеспечение синхронизации по приему и обработке данных, описание однозначно понимаемых и заказчиком, и исполнителем состояний автоматизируемого процесса – всё это возможно при использовании языка программирования SFC.

Основные достоинства SFC можно определить следующим образом:

– высокая выразительность. Язык SFC имеет те же возможности, что и диаграммы состояний, и является наиболее подходящим средством для описания динамических моделей;
– графическое представление. Благодаря графической мнемонике, SFC прост в использовании и изучении. Вместе с тем он является наглядным средством представления логики на разных уровнях детализации;
– предварительное проектирование ПО. Использование языка SFC на первых этапах проектирования прикладного ПО позволяет снять многочисленные непонимания между заказчиком, проектировщиком ПО и программистом.

Язык списка инструкций (IL)

До появления единого стандарта (до 1993 г.) почти каждый ПЛК сопровождался своим ассемблером. По этой причине целые поколения программистов были ориентированы на определенные серии микропроцессоров и микроконтроллеров. В итоге при освоении новой техники приходилось сталкиваться с проблемой освоения очередного языка программирования под новое устройство. Следует отметить, что отдельные конструкции ассемблеров были похожи, но всё же не было единого стандарта. Появление языка инструкций (Instruction List) как стандартного языка – это унификация интерфейса языка программирования низкого уровня, который не ориентирован на конкретную микропроцессорную архитектуру. Язык IL обладает весьма важным качеством: на его основе возможно создавать оптимальные по быстродействию программные единицы. Ниже продемонстрирован пример кода программы на языке списка инструкций:

```
1       LDN     I1
2       AN      I2
3       O       Q1
4       =       Q1
```

1.4.2. Среды программирования распространенных ПЛК

Codesys

CODESYS - (Controller Development System) – инструментальный программный комплекс промышленной автоматизации. Основой среды CODESYS является среда разработки прикладных программ для ПЛК.

Среда CODESYS поддерживает разработку на языках МЭК 61131-3: IL, LD, ST, FBD и SFC.

ABB

Control Builder F в первую очередь это среда разработки, и она же инструментальная система, оформленная как среда визуализации, т. е. по сути SCADA. В набор включен ABB OPC-Tunnel, средство разграничения доступа SecurityLock, сервер трендов TrendServer и средство CBF-Viewer.

Вся разработка происходит в среде Control Builder F. Здесь конфигурируется аппаратная часть, сеть, создаются и конфигурируются операторские станции, процессовые станции (т. е. контроллеры со своими входами-выходами), пишутся функциональные блоки, программмы, рисуются мнемосхемы и т. д.

Среда поддерживает разработку на языках МЭК 61131-3: IL, LD, ST, FB и SFC.

Schneider Electric

UnityPro, с 2019 г. UnityPro, начиная с версии 14.0 называется EcoStruxure Control Expert. Это среда программирования для ПЛК компании «Шнайдер Электрик». Она предоставляет инструменты для создания программ управления и настройки устройств «Шнайдер Электрик».

Среды программирования ПЛК «Шнайдер Электрик» поддерживают разработку на языках МЭК 61131-3: IL, LD, ST, FBD и SFC.

Siemens

TIA Portal (Totally Integrated Automation Portal) — среда разработки программного обеспечения для построения систем автоматизации технологических процессов различной степени сложности. Это среда программирования для ПЛК компании «Сименс». Она предоставляет инструменты для разработки программ управления и конфигурирования PLC и других компонентов автоматизации «Сименс».

Simatic Step 7 – программное обеспечение фирмы «Сименс» для разработки систем автоматизации на основе программируемых логических контроллеров с использованием языков программирования стандарта МЭК (LAD, FBD, STL). Применяется для программирования контроллеров S7-1200, S7-1500, S7-300, S7-400.

Simatic WinCC – приложение для разработки человеко-машинного интерфейса панелей оператора Simatic или SCADA.

Simatic PLCSIM – ПО, позволяющие эмулировать ПЛК необходимой конфигурации, HMI-панели или SCADA.

Среды программирования ПЛК «Сименс» поддерживают разработку на языках МЭК 61131-3: IL, LD, ST, FBD и SFC.

Rockwell Studio 5000

Среда программирования для ПЛК компании «Роквелл Автомейшн» позволяет разработчикам создавать программы управления, настраивать устройства и обеспечивать связь между ними.

Mitsubishi GX Works3

Среда программирования для ПЛК компании «Митсубиши Электрик» предоставляет инструменты для разработки программ управления и настройки ПЛК «Митсубиши Электрик».

Omron CX-One

Среда программирования для ПЛК компании «Омрон» позволяет разработчикам создавать программы управления, конфигурировать устройства и обеспечивать связь между ними.

Кроме того, существует множество других сред программирования для ПЛК, которые могут использоваться в зависимости от конкретных требований и предпочтений. К таким средам относятся B&R Automation Studio, Beckhoff TwinCAT и др.

1.5. Области применения ПЛК

Количество приложений ПЛК огромно. Согласно недавнему опросу, проведенному журналом Control Engineering: «Основные области применения ПЛК включают в себя управление машинами (87 %), управление процессами (58 %), управление движением (40 %), управление и контроль производственными операциями, которые выполняются последовательно (26 %), диагностику (18 %) и др. (3 %)». Результаты не дают в сумме 100 %, потому что одна система управления обычно имеет несколько приложений. ПЛК производятся и продаются по всему миру как отдельное оборудование несколькими крупными производителями систем автоматики. Кроме того, множество более специализированных компаний выпускают ПЛК для производителей оригинального оборудования (OEM).

ПЛК нашли широкое применение в различных отраслях, включая:

– производство – для управления многими процессами производства, такими как производство, монтаж и упаковка товаров. Они также ис-

пользуются для управления приводами, моторами, насосами и другими механизмами на заводах;

- энергетика – для управления различными системами электроэнергетической отрасли, такими как системы генерации, передачи и распределения электроэнергии для контроля за ними;
- автомобильная промышленность – для управления многими процессами в производстве автомобилей, такими как управление роботами, контроль качества, тестирование и сборка;
- машиностроение – для управления механизмами, такими как станки с ЧПУ, промышленные роботы, конвейеры и т. д.;
- обработка сигналов и автоматизация – для управления системами автоматизации и обработки сигналов, такими как системы управления доступом, системы безопасности и т. д.

В целом, ПЛК находят применение в любой отрасли, где требуется автоматизация и управление механизмами и процессами. Они обеспечивают надежность, гибкость и удобство управления в широком диапазоне промышленных и коммерческих приложений.

Конрольные темы для повторения

1. Типовая структура ПЛК. Элементы и их назначение.
2. Функциональные возможности ПЛК.
3. Стандартные языки программирования ПЛК МЭК 61131-3. Их описание, особенности.
4. Пример программы на стандартизированных языках программирования ПЛК МЭК 61131-3.
5. Общая информация о производителях ПЛК, среды программирования.

Глава 2. Основы логического управления

Дискретные входы и выходы (Digital Input/Output, DI/DO) – это основной тип входов и выходов, используемых в программируемых логических контроллерах. Дискретные входы могут быть использованы для получения дискретных сигналов от внешних устройств, таких как датчики и выключатели, а дискретные выходы могут быть использованы для управления такими устройствами, как клапаны, моторы и сигнальные средства.

Дискретные входы и выходы могут быть реализованы как встроенные в ПЛК модули и как внешние модули расширения. Они являются основой для многих автоматизированных систем управления и используются в широком спектре промышленных приложений, к примеру: управление движением конвейеров, управление температурой, системы контроля качества и т. д.

2.1. Описание дискретных входов

Дискретные входы – это тип входов, которые могут принимать только два состояния: «включено» или «выключено», – также известные как «1» или «0». Они представляют собой способ передачи дискретной информации в ПЛК от внешних устройств. Дискретные входы (рис. 2.1) могут быть использованы для обнаружения наличия или отсутствия сигнала, например для определения закрыта дверь или нет. Они могут также использоваться для измерения уровня напряжения, тока или частоты. Дискретные входы обычно обозначаются символом «I» – Input, за которой следует номер входа, например «I1» или «I2». Основным требованием к дискретным входам ПЛК является гальваническая развязка микроконтроллера от внешних устройств. Такая изоляция осуществляется при помощи специализированных микросхем – оптопар. Оптопары в своем составе содержат излучатель света (светодиод) и фотоприемник (на биполярных и полевых фототранзисторах, фотодиодах, фототиристорах, фоторезисторах), которые связаны оптическим каналом. Принцип работы оптопары заключается в преобразовании электрического сигнала в свет, передаче этого сигнала по оптическому каналу с последующим преобразованием его обратно в электрический сигнал.

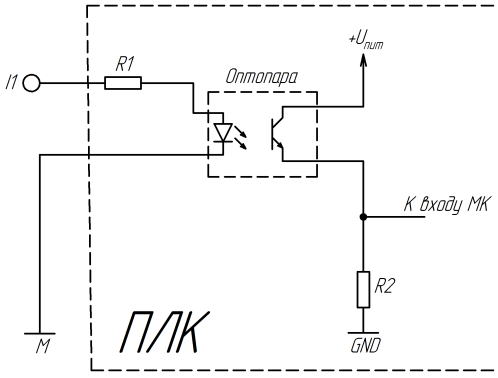


Рис. 2.1. Топология дискретного входа ПЛК

Важно отметить, что некоторые дискретные входы ПЛК могут быть чувствительны к полярности сигнала, который поступает на них. Это означает, что при изменении полярности сигнала, который подается на вход, его состояние может быть неправильно интерпретировано контроллером.

Для того чтобы обеспечить надежную работу дискретных входов при переполюсовке (рис. 2.2), производители ПЛК могут применять различные схемные решения. Одним из распространенных методов является использование специализированных оптопар с соединенными встречно-параллельно диодами (рис. 2.2, *а*). Эти оптопары позволяют эффективно изолировать дискретный вход от внешних воздействий и обеспечивать надежную работу в различных условиях полярности сигнала. Второй способ – применение мостового выпрямителя, установленного перед обычной оптопарой, (рис. 2.2, *б*).

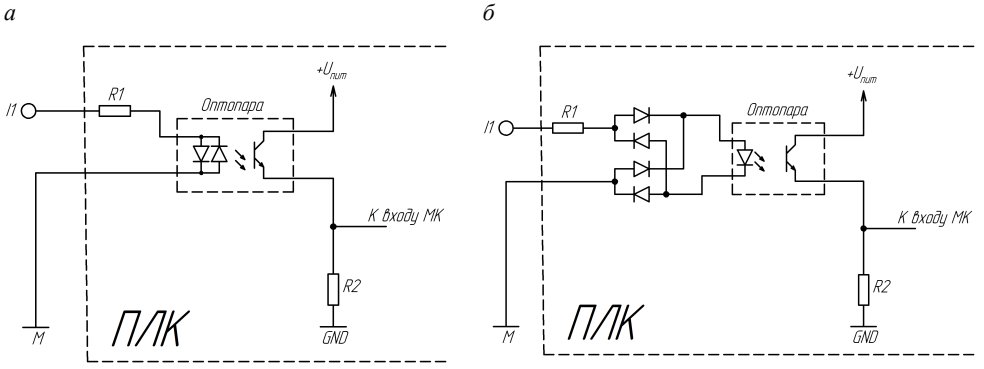


Рис. 2.2. Топологии дискретного входа ПЛК с возможностью переполюсовки с использованием специальных оптопар (*а*), с применением мостового выпрямителя (*б*)

2.2. Описание дискретных выходов

Дискретные выходы в ПЛК используются для управления внешними устройствами, такими как электромеханические реле, контакторы, соленоиды и др. Дискретные выходы ПЛК могут иметь два состояния: логический 0 (низкий уровень напряжения) или логическую 1 (высокий уровень напряжения).

Дискретные выходы могут использоваться для включения или выключения устройств или для создания импульсов управления с определенной длительностью. Например, дискретный выход может использоваться для включения и выключения света в комнате в зависимости от состояния датчика движения, который подключен к дискретному входу ПЛК. Дискретные выходы обычно обозначаются буквой «Q» - Quit, за которой следует номер входа, например «Q1» или «Q2».

2.2.1. Релейные выходы

Релейные выходы – это тип выходов, используемых в ПЛК и других автоматических системах управления. Они позволяют управлять внешними устройствами, такими как электромагнитные клапаны, световые индикаторы, электромоторы и др.

Релейные выходы могут быть реализованы с помощью механических реле или полупроводниковых реле (транзисторов), которые управляют выходными цепями и таким образом взаимодействуют с внешними устройствами. Релейные выходы могут быть настроены для переключения на высокий или низкий уровень напряжения в зависимости от значения входного сигнала.

Типовая топология релейного выхода (рис. 2.3) простая, дешевая, надежная. В этой схеме применение оптопар не требуется, т. к. в ней само реле изолирует выходы микроконтроллера от внешних устройств.

Для удержания реле малым током, с целью уменьшить нагрев внутренних цепей ПЛК, можно применять резистор с параллельно включенным конденсатором, установленным последовательно с катушкой реле (рис. 2.4, *а*). Если требуется быстрое отключение реле, то целесообразно применять стабилитрон, включенный последовательно с диодом, параллельно катушке реле (рис. 2.4, *б*).

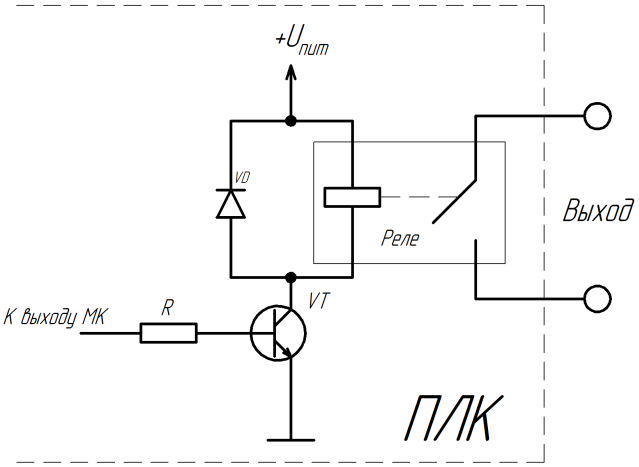


Рис. 2.3. Базовая топология релейного выхода ПЛК

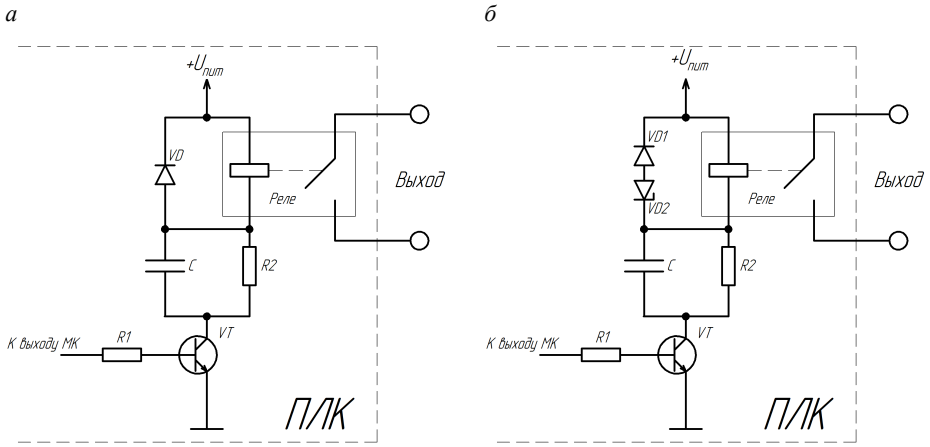


Рис. 2.4. Топологии модифицированных релейных выходов ПЛК с низким потреблением (*а*), с быстрым отключением реле (*б*)

2.2.2. Транзисторные выходы

Транзисторные дискретные выходы ПЛК – это электронные выходы, которые используются для управления электрическими устройствами в автоматизированных системах. Они позволяют ПЛК управлять электронными устройствами с высокой скоростью, а также имеют множество преимуществ по сравнению с релейными выходами.

В схемах этих выходов используются транзисторы в качестве ключевых элементов для управления электрическими нагрузками, такими как светодиоды, соленоиды, клапаны и др. [3]. Они могут иметь различные характеристики нагрузки в зависимости от производителя и модели ПЛК. Одним из главных преимуществ транзисторных дискретных выходов является их быстродействие, что позволяет им работать на высоких частотах переключения. Они также могут быть использованы для управления нагрузками с высоким током и мощностью, при этом занимая меньше места и потребляя меньше энергии, чем релейные выходы. Другим преимуществом транзисторных дискретных выходов является их долговечность. Транзисторы не имеют подвижных элементов, что уменьшает риск механического износа и повреждения, которые могут возникнуть при использовании релейных выходов. Однако у транзисторных дискретных выходов есть и некоторые ограничения. Например, они не могут работать с высокими напряжениями, их нельзя использовать для управления некоторыми типами нагрузок с переменным током, и они требуют специальной защиты от перенапряжений и коротких замыканий.

Транзисторные выходы могут быть PNP- и NPN-типа. Определяющим элементом в данном случае является тип транзистора, который используется при построении топологии. Следует отметить, что при построении транзисторных дискретных выходов требуется изоляция микроконтроллера от внешних устройств. Как и в случае с дискретными входами, для изоляции применяются оптопары. Топология транзисторного дискретного выхода NPN-типа показана на рис. 2.5. В данном случае при использовании нескольких дискретных выходов такого типа нужно помнить о том, что общая точка подключенных нагрузок будет присоединена к положительной клемме источника (VCC), питающего выходной каскад дискретных выходов.

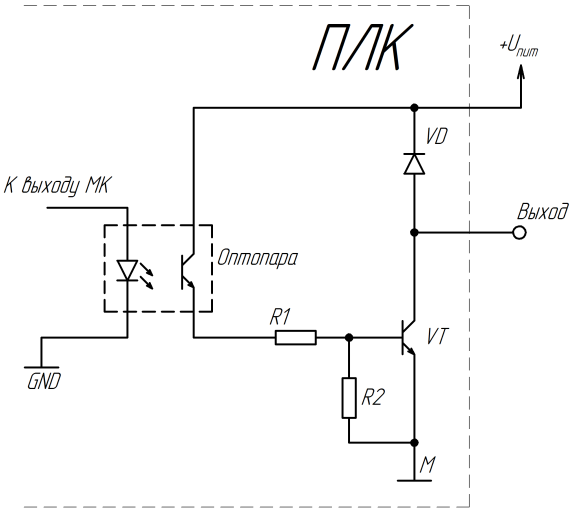


Рис. 2.5. Топология дискретного выхода NPN-типа

Подобную схему не всегда удобно использовать на практике, т. к. инженерам гораздо привычнее и удобнее подключать общую точку нагрузок к отрицательной клемме источника, питающего выходной каскад дискретных выходов. Решение, позволяющее реализовать данную концепцию, представлено на рис. 2.6 и реализовано при помощи транзисторов PNP-типа. В этой схеме общая точка нагрузок подключается к т. *M* источника, питающего выходной каскад дискретных выходов.

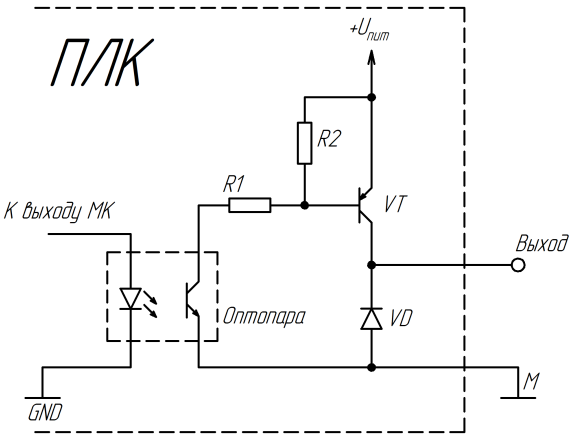


Рис. 2.6. Топология дискретного выхода PNP-типа

В представленных схемах решения по построению дискретных выходов показаны с применением биполярных транзисторов. Следует отметить, что подобные выходы возможно строить и с применением других типов транзисторов, принцип построения остаётся прежним: управление транзистором необходимо осуществлять при помощи оптопар.

2.3. Примеры программирования дискретных входов-выходов

2.3.1. Язык С

Для демонстрации примера программирования дискретных входов-выходов на языке С воспользуемся микроконтроллером Stm32. Микроконтроллеры фирмы ST являются весьма популярными у разработчиков электронных устройств. Для примера выбран микроконтроллер серии F0 [4] и среда программирования Stm32CubeIDE, предоставляемая фирмой-разработчиком микроконтроллеров [5]. Для упрощения процесса программирования, фирмой STMicroelectronics предлагается использовать библиотеку Hardware Abstraction Layer (HAL) [6]. Следует отметить, что представленный программный код с небольшими доработками применим для микроконтроллеров Stm32 различных серий.

Пример подключения и параметрирования пинов микроконтроллера PA0 в качестве дискретного входа и PB0, PB1, PB2 в качестве дискретных выходов показан ниже.

```
1  static void MX_GPIO_Init(void)
2  {
3      GPIO_InitTypeDef GPIO_InitStruct = {0};
4
5      /* GPIO Ports Clock Enable */
6      __HAL_RCC_GPIOA_CLK_ENABLE();
7      __HAL_RCC_GPIOB_CLK_ENABLE();
8
9      /* Configure GPIO pin Output Level */
```

```
10      HAL_GPIO_WritePin(GPIOB,
    ↪   GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2, GPIO_PIN_RESET);
11
12      /* Configure GPIO pin : PA0 */
13      GPIO_InitStruct.Pin = GPIO_PIN_0;
14      GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
15      GPIO_InitStruct.Pull = GPIO_PULLDOWN;
16      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
17
18      /* Configure GPIO pins : PB0 PB1 PB2 */
19      GPIO_InitStruct.Pin =
    ↪   GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2;
20      GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
21      GPIO_InitStruct.Pull = GPIO_PULLDOWN;
22      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
23      HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
24  }
```

Работа с дискретным входом и тремя дискретными выходами представлена в программном коде ниже. Алгоритм программы следующий.

Первым действием включаем дискретный выход PB0 при помощи функции **HAL_GPIO_WritePin(GPIOx, GPIO_Pin, PinState)**, аргументами которой являются имя порта, номер выхода, состояние вывода. Следующим этапом записываем в булеву переменную In0 состояние дискретнного входа PA0 при помощи функции **HAL_GPI_ReadPin(GPIOx, GPIO_Pin)**, аргументами которой являются имя порта и номер входа, и затем сотоянию выхода PB1 присваиваем значение переменной In0. Далее инвертируем выход PB2 при помощи функции **HAL_GPIO_TogglePin(GPIOx, GPIO_Pin)**, аргументами которой являются имя порта и номер выхода. Последним действием выключим дискретный выход PB0.

```
while (1)
{
    /*Включаем дискретный выход PB0*/
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    /*Записываем в булевую переменную <<In0>> состояние
    дискретнного входа PA0*/
    _Bool In0 = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
    /*In0 to PB1*/
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, In0);
    /*Инвертируем выход PB2*/
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_2);
    /*Выключаем дискретный выход PB0*/
   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
}
```

Для использования дискретных входов в режиме счетчика необходимо применить специализированную периферию – таймер, реализованный внутри микроконтроллера. Пример настройки таймера микроконтроллера Stm32F072CB представлен далее.

В данном примере используется таймер 3, который настраивается на работу в режиме счётчика импульсных сигналов. Пример демонстрирует подсчет импульсов инкрементального энкодера. Режим, в котором работает таймер, называется «Encoder Mode».

```
static void MX_TIM3_Init(void) {
    TIM_Encoder_InitTypeDef sConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload =
    TIM_AUTORELOAD_PRELOAD_DISABLE;
```

```
10      sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
11      sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
12      sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
13      sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
14      sConfig.IC1Filter = 15;
15      sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
16      sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
17      sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
18      sConfig.IC2Filter = 15;
19      if (HAL_TIM_Encoder_Init(&htim3, &sConfig) !=
    ↪   HAL_OK) {
20          Error_Handler();
21      }
22      sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
23      sMasterConfig.MasterSlaveMode =
    ↪   TIM_MASTERSLAVEMODE_DISABLE;
24      if (HAL_TIMEx_MasterConfigSynchronization(&htim3,
    ↪   &sMasterConfig) != HAL_OK) {
25          Error_Handler();
26      }
27  }
```

Для запуска таймера и прерываний, в которых будет выполняться программа, необходимо дать разрешение на запуск таймера и прерываний с помощью следующих команд.

```
1  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_1 |
    ↪   TIM_CHANNEL_2);
2  HAL_TIM_Encoder_Start_IT(&htim3, TIM_CHANNEL_1);
```

В прерывании таймера, когда приходят сигналы от инкрементального энкодера, для примера показано определение направления вращения энкодера, которое записывается в булеву переменную Dir, и количество подсчитанных импульсов, записанных в переменную Enc_Counter.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
  ↪  {
    if(htim->Instance == TIM3) {
        Dir = __HAL_TIM_IS_TIM_COUNTING_DOWN(&htim3);
        Enc_Counter = (TIM3->CNT);
    }
}
```

2.3.2. Язык LD

Поскольку по сравнению с языком C язык LD является языком более высокого уровня [7], в нём отсутствует возможность гибкого программирования. В данном случае пользователь может пользоваться только доступными ему командами. В примере программы для работы с дискретными входами-выходами демонстрируется следующий алгоритм (рис. 2.7).

Включается дискретный выход Q1, далее в переменную In0 записывается состояние дискретного входа I1, затем в дискретный выход Q2 записывается состояние переменной In0. На последнем шаге выключается выход Q3.

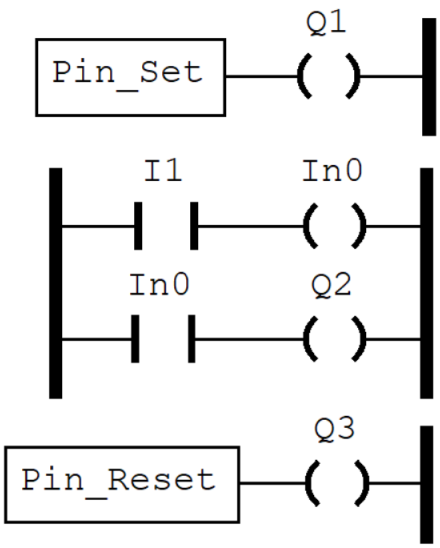


Рис. 2.7. Пример работы с дискретными входами-выходами на языке LD

2.3.3. Язык FBD

По сравнению с языком С, язык FBD является языком более высокого уровня, и в нем также отсутствует возможность гибкого программирования. Пользователь может пользоваться только доступными ему командами. В примере программы для работы с дискретными входами-выходами (рис. 2.8) демонстрируется применение функциональных блоков AND и OR, выполняющих логические функции «И» и «ИЛИ» соответственно. На вход функционального блока AND подаются сигналы от дискретных выходов I1 и I2, результат подается на дискретный выход Q1. На вход функционального блока OR подаются сигналы от дискретных выходов I3 и I4, результат подается на дискретный выход Q2.

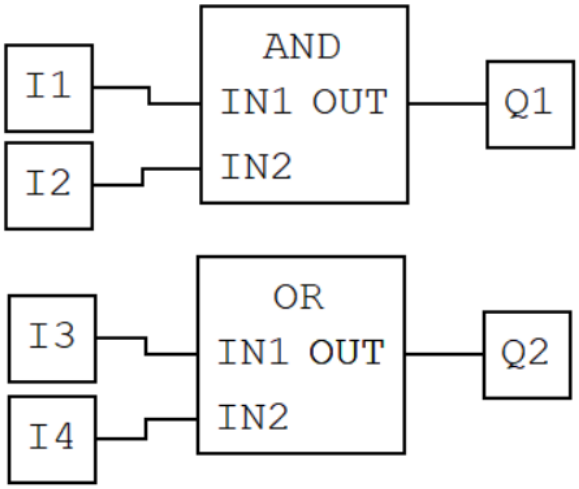


Рис. 2.8. Пример работы с дискретными входами-выходами на языке FBD

Конрольные темы для повторения

1. Дискретные входы ПЛК. Функции, описание.
2. Топология дискретных входов ПЛК.
3. Дискретные выходы ПЛК. Функции, описание.
4. Топология транзисторных дискретных входов ПЛК.
5. Топология релейных дискретных входов ПЛК.

Глава 3. Применение ПЛК в распределенных системах автоматизации

С ростом количества датчиков, увеличением площади, на которой расположена автоматизированная система, и с усложнением алгоритмов управления становится более эффективно применение распределенных систем. Распределенные системы состоят из множества удаленных друг от друга контроллеров и модулей ввода-вывода. В данном случае структура распределенной системы и структура алгоритма её работы подобны структуре самого объекта автоматизации, а функции сбора, обработки данных, управления и вычисления оказываются распределенными среди множества контроллеров. Каждый контроллер работает со своей группой устройств ввода-вывода и обслуживает определенную часть объекта управления. Кроме того, сосредоточенная система является частью или частным случаем распределенной, поэтому появление распределенных систем является следствием естественного развития от частного к общему.

Модель распределенной системы автоматизации в соответствии со стандартом МЭК 61499 может быть представлена как набор физических устройств (например, ПЛК), взаимодействующих между собой с помощью одной или нескольких промышленных сетей.

3.1. Сетевая модель OSI

Модель OSI (Open Systems Interconnection) — это концептуальная модель, используемая для описания сетевых протоколов и коммуникаций между компьютерными системами. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Она была разработана Международной организацией по стандартизации (ISO); она определяет семь уровней, каждый из них выполняет определенные функции и предоставляет определенные сервисы для обмена данными между узлами сети (данные на с. 39).

Уровни модели OSI

Название уровня	**Краткое описание**
7. Прикладной	Обеспечивает интерфейс для пользовательских приложений и предоставляет службы для приложений. Включает в себя протоколы для электронной почты, веб-браузеров, файловых передач и др.
6. Представления	Отвечает за представление данных, чтобы устройства с разными средствами представления могли взаимодействовать. Кодирует, сжимает, шифрует и форматирует данные для передачи
5. Сеансовый	Устанавливает, поддерживает и завершает сеансы связи (сессии) между устройствами в сети. Управляет установкой соединения, синхронизацией и сеансовыми параметрами
4. Транспортный	Обеспечивает надежную и точную доставку данных между узлами сети. Разбивает данные на меньшие блоки (сегменты), управляет потоком данных и обеспечивает контроль целостности и управление ошибками
3. Сетевой	Отвечает за маршрутизацию пакетов данных через сеть. Определяет логические адреса устройств (IP-адреса) и выбирает оптимальный путь для доставки данных
2. Канальный	Обеспечивает надежную передачу данных между двумя устройствами, связанными прямым физическим соединением или локальной сетью. Отправляет данные в виде кадров и обнаруживает и исправляет ошибки передачи
1. Физический	Занимается физическими аспектами передачи данных по сети, такими как электрические, оптические или радиочастотные сигналы. Определяет характеристики физических интерфейсов, таких как напряжение, скорость передачи и физический разъем

Модель OSI была разработана в кон. 1970-х гг. для поддержания разнообразных методов компьютерных сетей, которые в это время конкурировали за применение в крупных национальных сетевых взаимодействиях во Франции, Великобритании и США. В 1980-х гг. она стала рабочим продуктом группы взаимодействия открытых систем Международной организации по стандартизации (ISO). Модель не смогла дать полное описание сети и не получила поддержку архитекторов на заре интернета, который впоследствии нашел отражение в менее предписывающем TCP/IP, в основном под руководством Инженерного совета интернета (IETF).

Большинство сетей, в т. ч. промышленных, имеют далеко не все уровни модели OSI, но во всех сетях всегда есть первый, второй и, как правило, седьмой уровень.

3.2. Применяемые в промышленности цифровые сети

В промышленности широко используются различные цифровые сети и протоколы для обмена данными между устройствами и системами. Следует отметить такие протоколы в промышленной автоматизации, как протокол Modbus, Controller Area Network (CAN), Profibus и Industrial Ethernet.

Протокол ModBus является одним из наиболее распространенных протоколов в промышленной автоматизации. Он используется для связи между различными устройствами и системами управления, такими как ПЛК, датчики, исполнительные механизмы и др. Modbus может работать по различным физическим средам передачи данных, включая последовательные интерфейсы RS-232/485 и Ethernet.

CAN является широко применяемым протоколом и сетью в автомобильной и промышленной сферах. Он обеспечивает надежную передачу данных между устройствами в реальном времени. CAN позволяет соединять множество устройств в сеть и обмениваться сообщениями с высокой скоростью и отказоустойчивостью. CAN активно применяется для объединения промышленных устройств в единую сеть.

Протокол Profibus (Process Field Bus) используется для связи между различными устройствами автоматизации, такими как ПЛК, датчики, приводы и др. Он обеспечивает надежный и эффективный обмен данными в промыш-

ленных сетях. Profibus может работать как по последовательным интерфейсам, так и по промышленным Ethernet-сетям.

Протоколы серии Industrial Ethernet (например, EtherNet/IP, PROFINET, Modbus TCP) используют Ethernet-сеть для обмена данными в промышленных средах. Они предоставляют высокую пропускную способность, гибкость и отказоустойчивость, позволяя передавать данные в режиме реального времени и интегрировать различные устройства и системы автоматизации.

В данной работе, из-за широты применения в автоматизации зданий и сооружений, промышленности, предлагаются к рассмотрению протоколы Modbus и CAN.

3.2.1. ModBus

Протокол ModBus RTU (Remote Terminal Unit) был разработан компанией Modicon в 1979 г. для использования в ПЛК данного производителя. Протокол Modbus не полностью соответствует семиуровневой модели OSI, т. к. он был разработан до ее появления. Однако можно приблизительно соотнести протокол Modbus с физическим, канальным и прикладным уровнями модели OSI.

Основные преимущества протокола:

– открытость и отсутствие лицензирования – спецификация является общедоступной и выложена на официальном сайте ModBus [8];
– простота реализации – не требуются значительные аппаратные ресурсы, программная реализация также обычно является простой и компактной;
– распространенность – протокол поддерживается значительным количеством устройств.

К недостаткам протокола можно отнести:

– отсутствие типизации – данные передаются в виде битов или регистров. Передача данных с плавающей точкой, строк и т. д. не специфицирована и может отличаться у разных производителей. Зачастую это требует от пользователя низкоуровневой работы с памятью в программе ПЛК;
– отсутствие стандартизированного описания карт регистров в текстовом формате (например, .xml), что затрудняет настройку обмена с устройствами, т. к. пользователю требуется вручную прописывать параметры каждого запроса.

Физический уровень

Протокол Modbus определяет различные физические уровни в зависимости от используемой физической среды передачи данных. Вот несколько распространенных физических уровней для протокола Modbus:

– RS-232 (Recommended Standard 232) является стандартом для последовательной передачи данных между двумя устройствами через одиночный канал. Используется для Modbus RTU и работает на низкой скорости передачи данных (обычно 9 600 бит/с и 19 200 бит/с);
– RS-485 (Recommended Standard 485) является стандартом для полудуплексной последовательной передачи данных на длинных расстояниях. Используется для Modbus RTU, обеспечивая более высокую скорость передачи данных и возможность подключения нескольких устройств к одной линии. Использует дифференциальную передачу данных, что делает его более устойчивым к помехам;
– Ethernet. Modbus TCP является вариантом протокола Modbus, который работает поверх стандартного протокола Ethernet, а также поверх стека TCP/IP протоколов. В этом случае физический уровень представлен сетевыми интерфейсами, использующими Ethernet-кабели и сетевые разъемы.

Физический уровень Modbus зависит от конкретной реализации и требований системы. Он определяет физическую среду и характеристики передачи данных, которые необходимо учитывать при настройке и подключении устройств, работающих с протоколом Modbus.

Поскольку наиболее часто применяемым для организации промышленной сети является 2-проводный интерфейс RS-485, рассмотрим именно его [9].

Сеть, построенная на интерфейсе RS-485, представляет собой приемопередатчики, соединенные при помощи витой пары – двух скрученных проводов. В основе интерфейса RS-485 лежит принцип дифференциальной передачи данных (рис. 3.1). Суть его заключается в передаче одного сигнала по двум проводам, причем по одному проводу (условно А) идет оригинальный сигнал, а по другому (условно В) – его инверсная копия. Иначе говоря, если на одном проводе «1», то на другом «0», и наоборот. В результате между двумя проводами витой пары всегда есть разность потенциалов: при «1» она положительна, при «0» – отрицательна.

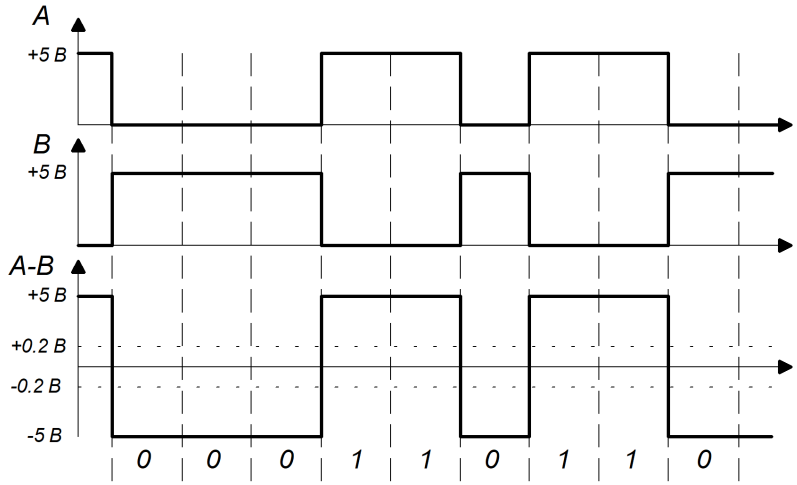


Рис. 3.1. График принципа дифференциальной передачи данных

Таким образом при помощи показанной разности потенциалов и передаётся сигнал. Подобный способ передачи обеспечивает высокую устойчивость к синфазной помехе, которая действует на оба провода линии одинаково. Например, электромагнитная волна, проходя через участок линии связи, наводит в обоих проводах некоторый потенциал. Если сигнал передается потенциалом в одном проводе относительно общего, как в RS-232, то наводка на этот провод может исказить сигнал относительно хорошо поглощающего наводки общего провода, имеющего условно нулевой потенциал («земли»). Кроме того, на сопротивлении длинного общего провода будет возникать разность потенциалов земель – дополнительный источник искажений. При дифференциальной передаче данных подобного искажения не происходит. Если два перевитых между собой провода пролегают близко друг к другу, то наводка на оба провода будет одинакова. Потенциал в обоих одинаково нагруженных проводах изменяется одинаково, при этом информативная разность потенциалов остается без изменений, как это показано на рис. 3.2.

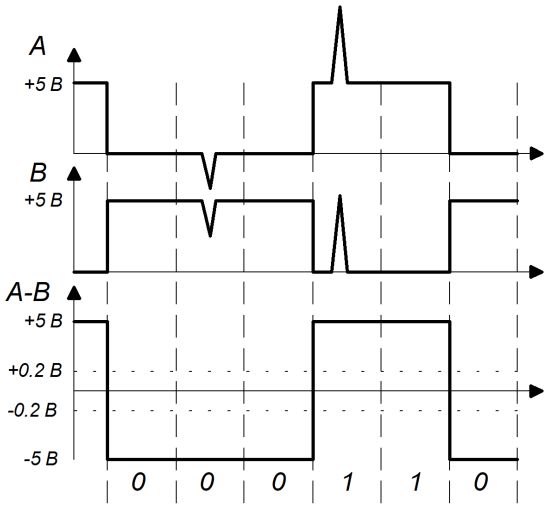


Рис. 3.2. Демонстрация устойчивости интерфейса RS-485 к синфазной помехе

Все устройства подключаются к одной витой паре одинаково: прямые выходы (*A*) к одному проводу, инверсные (*B*) – к другому, как показано на рис. 3.3.

В идеале Modbus-шина должна состоять из одного магистрального кабеля, от которого могут быть сделаны отводы. Магистральный кабель Modbus должен содержать 3 проводника в общем экране, два из которых представляют собой витую пару, а третий соединяет общие («земляные») выводы всех интерфейсов RS-485 в сети. Общий провод и экран должны быть заземлены в одной точке, желательно около ведущего устройства.

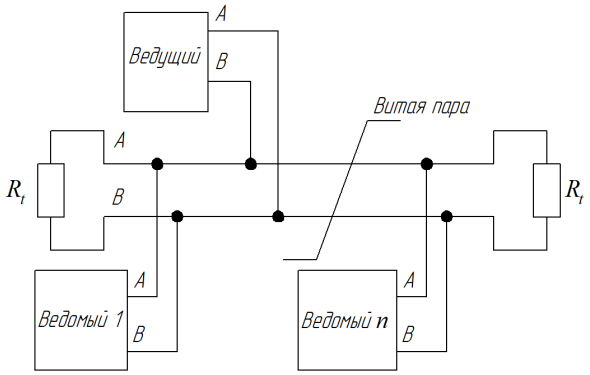


Рис. 3.3. Схема подключения шины Modbus

Устройства могут подключаться к кабелю тремя способами:

– непосредственно к магистральному кабелю,
– через пассивный разветвитель (тройник),
– через активный разветвитель, содержащий развязывающий повторитель интерфейса.

На каждом конце ModBus-шины должны быть установлены резисторы R_t (терминальные) для согласования линии передачи (рис. 3.4), как это требуется для интерфейса RS-485. Наличие данных резисторов в соответствии со стандартом Modbus обязательно и независимо от скорости обмена. Их номинал обычно выбирается равным 120 Ом при мощности 0,5 Вт. Терминальные резисторы, а также резисторы, устраняющие неопределённость состояния линии при высокоомном состоянии передатчиков, устанавливаются так же, как и в других сетях, на основе физического интерфейса RS-485. Стандарт требует, чтобы в руководствах по эксплуатации устройств Modbus было сказано, имеются ли указанные резисторы внутри устройства или их необходимо устанавливать при монтаже сети.

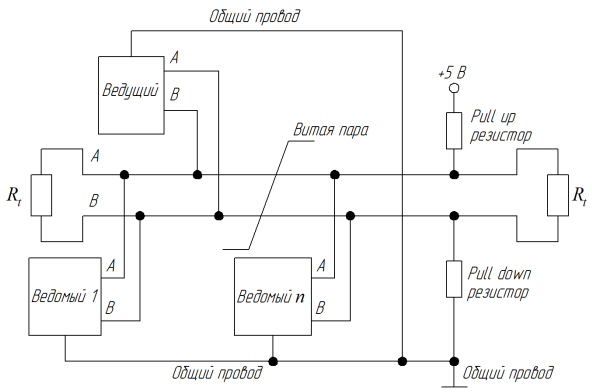


Рис. 3.4. Общая 2-проводная схема подключения шины Modbus

Аппаратная реализация интерфейса – микросхемы приемопередатчиков с дифференциальными входами-выходами, подключенными к линии, и цифровыми портами (к портам контроллера). RS-485 – полудуплексный интерфейс (рис. 3.5). Прием и передача идут по одной паре проводов с разделением по времени.

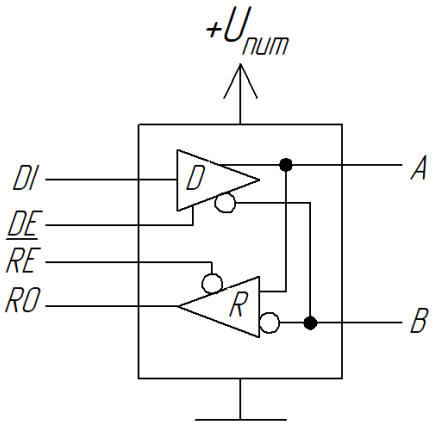


Рис. 3.5. Аппаратная реализация интерфейса RS-485

Приемник, получая на дифференциальных входах (A, B) разность потенциалов (U_{AB}), переводит их в цифровой сигнал на выходе RO. Чувствительность приемника может быть разной, но гарантированный пороговый диапазон напряжений распознавания сигнала прописан в документации на микросхему. Обычно эти пороги составляют ±200 мВ. Таким образом, если $U_{AB} > +200$ мВ, приемник определяет «1», а если $U_{AB} < -200$ мВ, приемник определяет «0». Если разность потенциалов в линии настолько мала, что не выходит за пороговые значения, не гарантируется правильное распознавание сигнала. Кроме того, в линии могут быть и несинфазные помехи, которые исказят столь слабый сигнал.

Modbus-устройство обязательно должно поддерживать скорости, бит/с: 9 600, 19 200, 38 400, 57 600, 115 200.

Скорость передачи должна выдерживаться в передатчике с погрешностью не хуже 1 %, а приёмник должен принимать данные при отклонении скорости передачи до 2 %.

Сегмент сети, не содержащий повторителей интерфейса, должен допускать подключение до 32 устройств, однако их количество может быть увеличено, если это допустимо, исходя из нагрузочной способности передатчиков и входного сопротивления приёмников, которые должны быть приведены в документации на интерфейсы. Указание этих параметров в документации является обязательным требованием стандарта.

Максимальная длина магистрального кабеля, при скорости передачи 9 600 бит/с и сечении жил более 0,13 мм^2 (AWG 26), составляет 1200 м. Отводы от магистрального кабеля не должны быть длиннее 20 м. При использовании многопортового пассивного разветвителя с N отводами, длина каждого отвода не должна превышать значения 40/N метров.

Канальный уровень

Канальный уровень Modbus выполняет функции кадрирования данных, обнаружения ошибок передачи данных, а также управления физическими параметрами связи [9].

Функция кадрирования данных означает, что каждое сообщение Modbus RTU имеет определенную структуру, состоящую из заголовка, полезных данных и контрольной суммы (рис. 3.6). Это позволяет получателю правильно интерпретировать передаваемые данные. На канальном уровне предусмотрены специфический для протокола формат кадров ADU (Application Data Unit). Прикладной уровень описывается форматом кадра PDU (Protocol Data Unit).

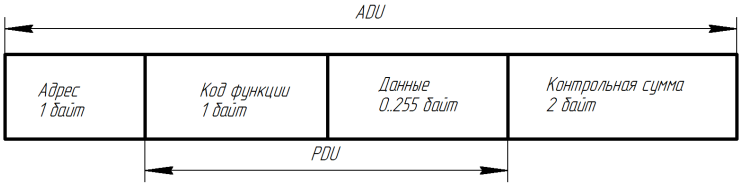


Рис. 3.6. Формат кадра ModBus

Протокол описывает детерминированный механизм доступа к среде по типу «ведущий – ведомый». Детерминированность в данном случае определяется тем, что в каждый момент времени активность на ModBus-шине определяется ведущим устройством. Оно формирует адресные запросы к ведомым устройствам и инициирует их на формирование пакета с ответными данными. В сети ModBus ведущим может выступать только одно устройство из всех присутствующих в любой момент времени.

Ведущее устройство может формировать запросы к ведомому устройству. Ведомое устройство отвечает на запросы ведущего. Инициатором обмена всегда выступает ведущее устройство. По указанной причине ведомое устройство не может передавать данные по мере их появления, поэтому в сети ModBus требуется интенсивный опрос ведомых устройств ведущим.

В ModBus RTU каждый байт сообщения содержит два 4-битных шестнадцатиричных числа. Каждое сообщение передается непрерывным потоком. Каждый байт данных передаётся младшими разрядами вперёд. Байт начинается со старт-бита, после которого следуют 8 бит данных, бит четности и стоп-бит (рис. 3.7). Таким образом, одна посылка данных состоит из 11 бит.

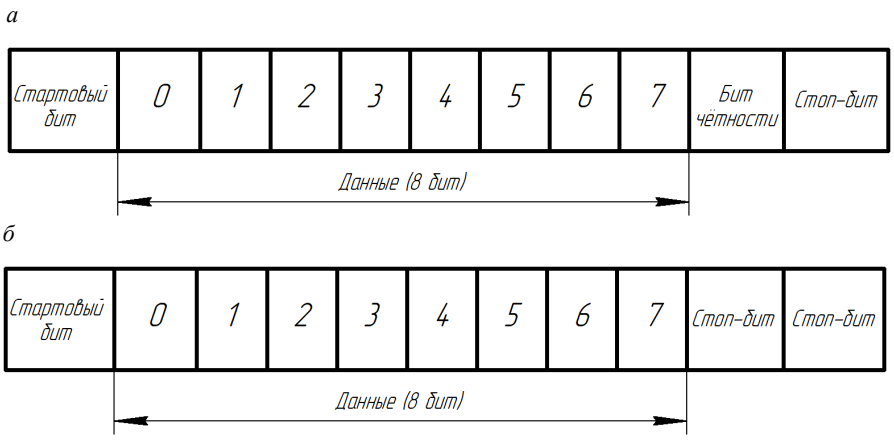


Рис. 3.7. Формат байта ModBus с битом четности *(а)* и без бита чётности *(б)*

По умолчанию в режиме RTU бит паритета устанавливают равным 1, если количество двоичных единиц в байте нечетное, и равным 0, если оно четное. Такой паритет называют четным (even parity), а метод контроля называют контролем паритета. При четном количестве двоичных единиц в байте, бит паритета может быть равен 1. В этом случае паритет называется нечетным (odd parity). Для согласования контроллера с различными устройствами присутствует возможность работы без бита четности, но в этом случае используется два стоп-бита. Пользователь может настраивать контроллеры на проверку четного или нечетного паритета (Even/Odd). Например, 8 бит RTU-режима содержат следующую информацию: 1100 0101. Общее количество единиц – 4. Если используется четный паритет, то бит паритета будет равен 0 и общее количество 1 будет по-прежнему четным числом. Если используется нечетный паритет, то бит паритета будет равен 1, тогда общее количество единиц вместе с битом паритета будет равно 5, т. е. нечетному числу.

Обнаружение ошибок передачи данных включает в себя проверку целостности данных с помощью контрольной суммы и проверку паритета байта. Если при получении данных обнаруживается ошибка, принимающая сторона может запросить повторную передачу данных.

В RTU-режиме сообщение начинается с интервала тишины, равного времени передачи 3,5 символов, при выбранной скорости передачи в сети. Первым полем передается адрес устройства. Вслед за последним передаваемым символом следует интервал тишины продолжительностью не менее 3,5 символов. Новое сообщение может начинаться после этого интервала (рис. 3.8).

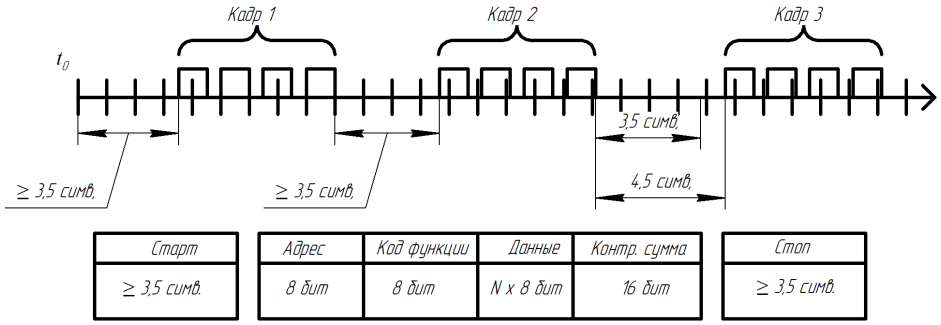


Рис. 3.8. Временная диаграмма передаваемого кадра в сети ModBus RTU

Кадр сообщения передается непрерывно. Если интервал тишины продолжительностью более 1,5 символа возник во время передачи кадра, принимающее устройство заканчивает прием сообщения, кадр сообщения считается неполным и следующий байт будет воспринят как начало следующего кадра. Пример показан на рис. 3.9.

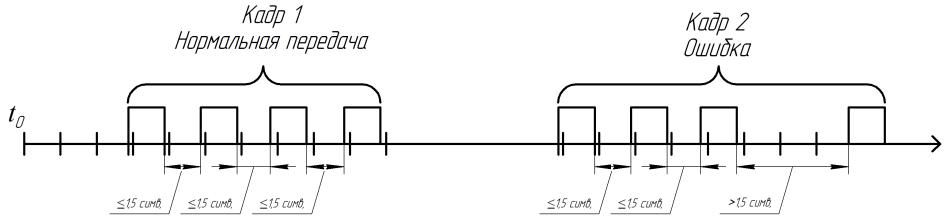


Рис. 3.9. Временная диаграмма нормальной передачи кадра и передачи кадра с ошибкой в сети ModBus RTU

Адресное поле кадра состоит из 8 бит. Адреса 1–247 – адреса Modbus-устройств в сети, адреса 248–255 являются зарезервированными. Ведущее устройство не имеет своего адреса, также следует помнить, что в сети не должно быть двух устройств с одинаковыми адресами. Ведущее устройство может посылать запросы всем устройствам одновременно (широковещательный режим) или только одному. Для широковещательного режима зарезервирован адрес 0. При использовании в команде этого адреса, она принимается всеми устройствами сети, которые не посылают ведущему устройству ответ.

Поле кода функции содержит 8 бит. Диапазон числа 1–255. Когда ведомое устройство отвечает ведущему, оно использует поле кода функции для фиксации ошибки. В случае нормального ответа ведомое устройство повторяет оригинальный код функции. Если имеет место ошибка, код функции возвращается с установленным в 1 старшим битом.

Поле данных в сообщении от ведущего устройства к ведомому содержит дополнительную информацию, которая необходима ведомому для выполнения указанной функции. Оно может содержать адреса регистров или выходов, их количество, счетчик передаваемых байтов данных.

Контрольная сумма CRC (Cyclic Redundancy Check), предназначенная для проверки целостности данных, состоит из двух байтов. Контрольная сумма вычисляется передающим устройством и добавляется в конец сообщения. Принимающее устройство вычисляет контрольную сумму в процессе приема и сравнивает ее с полем CRC принятого сообщения. Только 8 бит данных используются для вычисления контрольной суммы CRC. Старт и стоп-биты, бит паритета, если он используется, не учитываются в контрольной сумме.

Прикладной уровень

Прикладной уровень Modbus RTU обеспечивает коммуникацию между устройствами типа «ведущий – ведомый». Прикладной уровень является независимым от физического и канального, в частности он может использовать протоколы Ethernet TCP/IP (Modbus TCP/IP), интерфейсы RS-232, RS-422, RS-485 [10].

Прикладной уровень Modbus основан на запросах с помощью кодов функций. Код функции указывает ведомому устройству, какую операцию оно должно выполнить.

Обмен данными с устройствами происходит через регистры. Протоколом Modbus определяется четыре типа регистров (табл. 1).

Регистры дискретных выходов (Coils), хранящие однобитные значения, могут находиться в состоянии «0» или «1». Такие регистры могут обозначать текущее состояние выхода. «Coil» и означает катушку электромеханического реле. Данные регистры допускают как чтение, так и запись.

Таблица 1

Модель данных Modbus

Регистры	**Разрядность**	**Доступ**
Дискретные выходы (Coils)	1 бит	Чтение и запись
Дискретные входы (Discrete Inputs)	1 бит	Только чтение
Регистры хранения (Holding Registers)	16-битное слово	Чтение и запись
Входные регистры (Input Registers)	16-битное слово	Только чтение

Дискретные входы (Discrete Inputs) являются однобитными регистрами, которые описывают состояния входов устройства (например, при поданном напряжении, бит устанавливается в «1»). Эти регистры поддерживают только чтение.

Регистры хранения (Holding Registers) и входные регистры (Input Registers) представлены двухбайтовым словом и могут хранить значения 0–65535 (0x0000…0xFFFF). Входные регистры допускают только чтение (например, текущее значение на аналоговом входе). Регистры хранения поддерживают как чтение, так и запись (для хранения информации).

При использовании протокола прикладного уровня с различными протоколами транспортного и канального уровня сохраняется неизменным основной блок Modbus-сообщения, включающий код функции и данные, (этот блок называется PDU – Protocol Data Unit – элемент данных протокола). К блоку PDU могут добавляться дополнительные поля при использовании его в различных

промышленных сетях, и тогда его называют ADU – Application Data Unit – элемент данных приложения.

Стандартом Modbus предусмотрено три категории кодов функций: установленные стандартом, задаваемые пользователем и зарезервированные.

Коды функций – это числа в диапазоне 1–127. Диапазон 65–72 и 100–110 отвечает за задаваемые пользователем функции, диапазон 128–255 зарезервирован для передачи кодов ошибок в ответном сообщении. Код функции 0 не используется.

Коды ошибок используются ведомым устройством для определения действия, которое необходимо предпринять для их обработки. Значения кодов с их описанием изложены в стандарте Modbus RTU [10].

Поле данных (см. рис. 3.6) сообщения, посланного от ведущего устройства ведомому, содержит информацию, которую ведомое устройство использует, чтобы выполнить функцию, указанную в поле «Код функции». Поле данных может содержать значения состояний дискретных входов-выходов, адреса регистров, из которых надо считывать (записывать) данные, количество байтов данных, ссылки на переменные, количество переменных и др.

Если ведомое устройство корректно выполнило принятую от ведущего устройства функцию, то в ответе поле «Код функции» содержит ту же информацию, что и в запросе. В противном случае ведомый выдает код ошибки. В случае ошибки код функции в ответе равен коду функции в запросе, увеличенному на 128. Например, сообщение от ведущего устройства подчиненному посылает прочитать группу регистров и имеет следующий код функции: 0000 0011 (0x03). Если ведомое устройство выполнило затребованное действие без ошибки, оно возвращает такой же код. Если имеет место ошибка, то оно возвращает: 1000 0011 (0x83). В добавление к изменению кода функции, ведомое устройство размещает в поле данных уникальный код, который указывает ведущему, какая именно ошибка произошла, или на причину ошибки.

В сообщении ведущего устройства ведомому поле данных содержит дополнительную информацию, необходимую для выполнения указанной функции. Конкретное содержание поля данных устанавливается стандартом для каждой функции отдельно. В некоторых сообщениях поле данных может иметь нулевую длину.

Далее представлены распространённые коды функций. В качестве примеров показана структура PDU для каждого из кодов функций.

01 (0x01) Read Coils

Функция с кодом 01 (или 0x01 в шестнадцатеричной системе) используется для считывания состояний дискретных выходов в удаленном устройстве. Этот запрос позволяет считать 1–2000 последовательных состояний дискретных выходов.

В запросе передается начальный адрес, т. е. адрес первого указанного дискретного выхода, и количество выходов, которое необходимо считать. В запросе дискретные выходы нумеруются с 0, таким образом, дискретные выходы с номерами 1–16 будут адресованы как 0–15.

В ответе упакованные данные представляются в виде одной катушки на 2 бита поля данных. Состояния дискретных выходов отображаются как 1 (включено) или 0 (выключено). Например, младший бит первого байта данных содержит состояние выхода, адресованного в запросе. Последующие катушки следуют по направлению к старшему концу этого байта и далее, от младшего к старшему, в последующих байтах.

Если количество возвращенных выходных данных не кратно восьми, оставшиеся биты в последнем байте данных будут заполнены нулями (в сторону старшего конца байта). В поле «Количество байт данных» указывается количество полных байтов данных.

Структура PDU запроса и ответа для кода функции 0x01 (считывание состояний дискретных выходов) приведена в табл. 2.

В табл. 3 показан пример запроса и ответа на чтение состояния дискретных выходов 20–38.

Таблица 2

Структура PDU запроса и ответов кода функции 0x01

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x01
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во дискретных выходов	2	1 to 2000 (0x7D0)
Ответ (нормальный)		
Код функции	1	0x01
Кол-во байтов данных	1	**N***
Сост. дискретных выходов	*n* байтов	n = N или N+1
Ответ (ошибка)		
Код функции	1	Код ф-ции + 0x80
Код ошибки	1	0x01…0x04

***N** = Количество дискретных выходов / 8; если остаток от деления отличен от 0, то N = N + 1.

Таблица 3

Пример PDU запроса и ответа кода функции 0x01

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x01	Код функции	0x01
Начальный адрес (Hi)	0x00	Кол-во байтов данных	0x03
Начальный адрес (Lo)	0x13	Сост. дискр. вых. (27-20)	0xCD
Кол-во дискр. вых. (Hi)	0x00	Сост. дискр. вых. (35-28)	0x6B
Кол-во дискр. вых. (Lo)	0x13	Сост. дискр. вых. (38-36)	0x05

Состояние выходов 27–20 отображается в виде значения байта 0xCD (в двоичной форме – 1100 1101). Выход 27 соответствует старшему биту этого байта, а выход 20 – младшему биту. По соглашению протокола ModBus RTU, биты внутри байта отображаются с наиболее значимым битом слева и наименее значимым битом справа. Таким образом, выходы 27–20 находятся в первом байте данных, слева направо. Следующий байт содержит состояние выходов 35–28, также расположенных слева направо.

Процесс продолжается, и каждый последующий байт содержит состояния выходов от младшего к старшему, например: 20…27, 28…35 и т. д. В последнем байте данных состояние выходов 38–36 представлено значением 0x05 (в двоичной форме – 0000 0101). Выход 38 находится в шестом битовом положении слева, а выход 36 – в младшем бите этого байта. Пять оставшихся старших битов заполнены нулями.

02 (0x02) Read Discrete Inputs

Данный код функции используется для считывания 1–2000 непрерывных состояний дискретных входов в удаленном устройстве. PDU запроса указывает начальный адрес, т. е. адрес первого заданного входа, и количество входов. В дискретных входах PDU-адресация начинается с нуля, поэтому дискретные входы с номерами 1–16 адресуются как 0–15.

Дискретные входы в ответном сообщении упаковываются как один вход на бит поля данных. Статус отображается как 1 = ВКЛ, 0 = ВЫКЛ. Младший бит первого байта данных содержит вход, адресованный в запросе. Другие входы следуют к старшему концу этого байта и от младшего к старшему в последующих байтах.

Если возвращенное входное количество не кратно восьми, оставшиеся биты в последнем байте данных будут дополнены нулями (в сторону старшего конца байта). Поле «Количество байт данных» указывает количество полных байтов данных. Структура PDU запроса и ответов кода функции 0x02 для чтения состояний дискретных входов показана в табл. 4.

Таблица 4

Структура PDU запроса и ответов кода функции 0x02

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x02
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во дискретных входов	2	1 to 2000 (0x7D0)
Ответ (нормальный)		
Код функции	1	0x02
Кол-во байтов данных	1	**N***
Сост. дискретных входов	**N*** × 1 байт	
Ответ (ошибка)		
Код функции	1	0x82
Код ошибки	1	0x01…0x04

***N** = количество дискретных входов / 8; если остаток от деления отличен от 0, то N = N + 1.

В табл. 5 показан пример запроса и ответа на чтение состояния дискретных входов 197–218.

Таблица 5

Пример PDU запроса и ответа кода функции 0x02

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x02	Код функции	0x02
Начальный адрес (Hi)	0x00	Кол-во байтов данных	0x03
Начальный адрес (Lo)	0xC4	Сост. дискр. вх. (204-197)	0xAC
Кол-во дискр. вх. (Hi)	0x00	Сост. дискр. вх. (212-205)	0xDB
Кол-во дискр. вх. (Lo)	0x16	Сост. дискр. вх. (218-213)	0x35

Состояние дискретных входов 204–197 отображается как значение байта 0xAC (в двоичном виде – 1010 1100). Вход 204 – это старший бит этого байта, а вход 197 – младший бит.

Состояние дискретных входов 218–213 отображается как значение байта 0x35 (в двоичном виде – 0011 0101). Вход 218 находится в третьем разряде слева, а вход 213 является младшим битом.

03 (0x03) Read Holding Registers

Код функции используется для чтения содержимого регистров хранения данных в удаленном устройстве. В PDU запроса определяется адрес начального регистра и количество регистров. В PDU регистры адресуются начиная с нуля, поэтому регистры с номерами 1–16 адресуются как 0–15.

Данные регистра в ответном сообщении упакованы по два байта на регистр, при этом двоичное содержимое выравнивается по правому краю в каждом байте. Для каждого регистра первый байт содержит старшие биты, а второй – содержит младшие биты. Структура PDU запроса и ответов кода функции 0x03 для чтения содержимого регистров хранения данных показана в табл. 6.

Таблица 6

Структура PDU запроса и ответов кода функции 0x03

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x03
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во регистров	2	1 to 125 (0x7D)
Ответ (нормальный)		
Код функции	1	0x03
Кол-во байтов данных	1	2 × **N***
Значения регистров	**N*** × 2 байта	
Ответ (ошибка)		
Код функции	1	0x83
Код ошибки	1	0x01…0x04

***N** = количество регистров.

В табл. 7 показан пример запроса и ответа на чтение регистров 108–110.

Содержимое регистра 108 отображается в виде двухбайтовых значений 0x022B или 555 в десятичном виде. Содержимое регистров 109–110 – 0x0000 и 0x0064 или 0 и 100 в десятичном виде соответственно.

Таблица 7

Пример PDU запроса и ответа кода функции 0x03

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x03	Код функции	0x03
Начальный адрес (Hi)	0x00	Кол-во байтов данных	0x06
Начальный адрес (Lo)	0x6B	Знач. регистра 108 (Hi)	0x02
Кол-во регистров (Hi)	0x00	Знач. регистра 108 (Lo)	0x2B
Кол-во регистров (Lo)	0x3	Знач. регистра 109 (Hi)	0x00
–	–	Знач. регистра 109 (Lo)	0x00
–	–	Знач. регистра 110 (Hi)	0x00
–	–	Знач. регистра 110 (Lo)	0x64

04 (0x04) Read Input Registers

Данный код функции используется для чтения 1–125 входных регистров в удаленном устройстве. PDU запроса определяет адрес начального регистра и количество регистров. В PDU регистры адресуются начиная с нуля, поэтому входные регистры с номерами 1–16 адресуются как 0–15.

Данные регистра в ответном сообщении упакованы по два байта на регистр, при этом двоичное содержимое выравнивается по правому краю в каждом байте. Для каждого регистра первый байт содержит старшие биты, а второй – содержит младшие биты. Структура PDU запроса и ответов кода функции 0x04 для чтения входных регистров показана в табл. 8.

Таблица 8

Структура PDU запроса и ответов кода функции 0x04

Обозначение	**Кол-во байт**	**Значение**
Запрос		
Код функции	1	0x04
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во входн. регистров	2	0x0001 to 0x007D
Ответ (нормальный)		
Код функции	1	0x04
Кол-во байтов данных	1	2 × **N***
Сост. входн. регистров	**N*** × 2 байта	
Ответ (ошибка)		
Код функции	1	0x84
Код ошибки	1	0x01…0x04

***N** = количество регистров.

В табл. 9 показан пример запроса и ответа на чтение входного регистра 9.

Содержимое входного регистра 9 отображается в виде двухбайтовых значений 0x000A в шестнадцатеричном виде или 10 в десятичном виде.

Таблица 9

Пример PDU запроса и ответа кода функции 0x04

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x04	Код функции	0x04
Начальный адрес (Hi)	0x00	Кол-во байтов данных	0x02
Начальный адрес (Lo)	0x08	Знач. входн. рег. 9 (Hi)	0x00
Кол-во входн. рег. (Hi)	0x00	Знач. вход. рег. 9 (Lo)	0x0A
Кол-во входн. регистров (Lo)	0x01	–	–

05 (0x05) Write Single Coil

Функциональный код используется для записи состояния одного дискретного выхода на удаленном устройстве в положение ВКЛ (включено) или ВЫКЛ (выключено). Запрашиваемое состояние ВКЛ/ВЫКЛ задается постоянным значением в поле данных запроса. Значение 0xFF00 используется для включения выхода, а значение 0x0000 – для выключения дискретного выхода. Все другие значения недопустимы.

В PDU запроса указывается адрес дискретного выхода, который должен быть переключен. Дискретные выходы адресуются начиная с нуля, поэтому дискретный выход с номером 1 соответствует адресу 0. Обычный ответ – это эхо запроса, возвращаемое после записи состояния дискретного выхода. Структура PDU запроса и ответов кода функции 0x05 для записи одного дискретного выхода в положение ВКЛ или ВЫКЛ показана в табл. 10.

В табл. 11 показан пример запроса и ответа на запись в дискретный выход 173 значения ВКЛ.

Таблица 10

Структура PDU запроса и ответов кода функции 0x05

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x05
Адрес дискр. выхода	2	0x0000 to 0xFFFF
Значение для записи	2	0x0000 or 0xFF00
Ответ (нормальный)		
Код функции	1	0x05
Адрес дискр. выхода	2	0x0000 to 0xFFFF
Значение для записи	2	0x0000 or 0xFF00
Ответ (ошибка)		
Код функции	1	0x85
Код ошибки	1	0x01…0x04

Таблица 11

Пример PDU запроса и ответа кода функции 0x05

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x05	Код функции	0x05
Адрес дискр. выхода (Hi)	0x00	Адрес дискр. выхода (Hi)	0x00
Адрес дискр. выхода (Lo)	0xAC	Адрес дискр. выхода (Lo)	0xAC
Значение для записи (Hi)	0xFF	Значение для записи (Hi)	0xFF
Значение для записи (Lo)	0x00	Значение для записи (Lo)	0x00

06 (0x06) Write Single Register

Код функции используется для записи одного регистра хранения данных в удаленном устройстве. PDU запроса определяет адрес записываемого регистра. Регистры адресуются начиная с нуля, поэтому регистр с номером 1 адресуется как 0. Обычный ответ — это эхо запроса, возвращаемое после того, как содержимое регистра было записано. Структура PDU запроса и ответов кода функции 0x06 для записи одного регистра хранения данных показана в табл. 12. В табл. 13 показан пример запроса и ответа на запись в регистр 2 значения 0x0003.

Таблица 12

Структура PDU запроса и ответов кода функции 0x06

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x06
Адрес регистра	2	0x0000 to 0xFFFF
Значение для записи	2	0x0000 to 0xFFFF
Ответ (нормальный)		
Код функции	1	0x06
Адрес регистра	2	0x0000 to 0xFFFF
Значение для записи	2	0x0000 or 0xFFFF
Ответ (ошибка)		
Код функции	1	0x86
Код ошибки	1	0x01…0x04

Таблица 13

Пример PDU запроса и ответа кода функции 0x06

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x06	Код функции	0x06
Адрес регистра (Hi)	0x00	Адрес регистра (Hi)	0x00
Адрес регистра (Lo)	0x01	Адрес регистра (Lo)	0x01
Значение для записи (Hi)	0x00	Значение для записи (Hi)	0x00
Значение для записи (Lo)	0x03	Значение для записи (Lo)	0x03

15 (0x0F) Write Multiple Coils

Данный код функции используется для включения или выключения каждого дискретного выхода в последовательности дискретных выходов на удаленном устройстве. PDU запроса определяет адреса дискретных выходов, которые должны быть переключены. Дискретные выходы адресуются начиная с нуля, поэтому дискретный выход с номером 1 соответствует адресу 0.

Состояния ВКЛ/ВЫКЛ определяются содержимым поля данных запроса. Логическая «1» в битовой позиции поля требует включения соответствующего выхода. Логический «0» требует выключения.

Обычный ответ возвращает код функции, начальный адрес и количество переключенных катушек. Структура PDU запроса и ответов кода функции 0x0F для записи одного регистра хранения данных показана в табл. 14.

В табл. 15 показан пример запроса и ответа на запись состояния 10 дискретных выходов, начиная с 20-го.

Таблица 14

Структура PDU запроса и ответов кода функции 0x0F

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x0F
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во дискр. выходов	2	0x0001 to 0x07B0
Количество байтов	1	**N***
Значение для записи	**N*** × 1 байт	
Ответ (нормальный)		
Код функции	1	0x0F
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во дискр. выходов	2	0x0001 to 0x07B0
Ответ (ошибка)		
Код функции	1	0x8F
Код ошибки	1	0x01…0x04

***N** = Кол-во дискретных выходов / 8, если остаток от деления не 0, то N = N+1.

Таблица 15

Пример PDU запроса и ответа кода функции 0x0F

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x0F	Код функции	0x0F
Нач. адр. дискр. вых. (Hi)	0x00	Нач. адр. дискр. вых. (Hi)	0x00
Нач. адр. дискр. вых. (Lo)	0x13	Нач. адр. дискр. вых. (Lo)	0x13
Кол-во дискр. вых. (Hi)	0x00	Кол-во дискр. вых. (Hi)	0x00
Кол-во дискр. вых. (Lo)	0x0A	Кол-во дискр. вых. (Lo)	0x0A
Количество байтов	0x02	–	–
Значение для записи (Hi)	0xCD	–	–
Значение для записи (Lo)	0x01	–	–

Содержимое поля «Значение для записи» состоит из двух байтов: 0xCD01 (в двоичном виде – 1100 1101 0000 0001). Первый переданный байт (0xCD) указывает на дискретные выходы 27–20, причем младшим значащим битом адресуется младший выход (20) в этом наборе.

Следующий передаваемый байт (0x01) указывает на выходы 29–28, причём младшим значащим битом адресует младший выход (28) в этом наборе. Неиспользуемые биты в последнем байте данных заполняются нулями.

16 (0x10) Write Multiple registers

Код функции используется для записи последовательности регистров хранения данных (1–123 регистров) в удаленном устройстве.

Запрашиваемые записываемые значения указываются в поле данных запроса. Данные упакованы как два байта на регистр. Обычный ответ возвращает код функции, начальный адрес и количество записанных регистров.

Структура PDU запроса и ответов кода функции 0x10 для записи последовательности регистров хранения данных показана в табл. 16.

Таблица 16

Структура PDU запроса и ответов кода функции 0x10

Обозначение	**Кол-во байтов**	**Значение**
Запрос		
Код функции	1	0x10
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во регистров	2	0x0001 to 0x007B
Количество байтов	1	2 × **N***
Значение для записи	**N*** × 2 байта	Значение
Ответ (нормальный)		
Код функции	1	0x10
Начальный адрес	2	0x0000 to 0xFFFF
Кол-во регистров	2	1 to 123 (0x7B)
Ответ (ошибка)		
Код функции	1	0x90
Код ошибки	1	0x01…0x04

***N** = Количество регистров хранения данных.

В табл. 17 показан пример запроса и ответа на запись в два регистра хранения данных, начиная со второго, значений 0x000A и 0x0102 соответственно.

Команда состоит из начального адреса 0x0001, количества изменяемых элементов 0x0002, количества передаваемых байтов устанавливаемых значений 0x04 и самих устанавливаемых значений 0x000A и 0x0102.

Ответ состоит из начального адреса 0x0001 и количества изменённых элементов 0x0002.

Таблица 17

Пример PDU запроса и ответа кода функции 0x10

Запрос		***Ответ***	
Обозначение	**Значение**	**Обозначение**	**Значение**
Код функции	0x10	Код функции	0x10
Нач. адрес (Hi)	0x00	Нач. адрес (Hi)	0x00
Нач. адрес (Lo)	0x01	Нач. адрес (Lo)	0x01
Кол-во регистров (Hi)	0x00	Кол-во регистров (Hi)	0x00
Кол-во регистров (Lo)	0x02	Кол-во регистров (Lo)	0x02
Количество байтов	0x04	–	–
Знач. рег. для записи (Hi)	0x00	–	–
Знач. рег. для записи (Lo)	0x0A	–	–
Знач. рег. для записи (Hi)	0x01	–	–
Знач. рег. для записи (Lo)	0x02	–	–

Коды ошибок ModBus

В табл. 18 приведён список стандартных кодов ошибок протокола ModBus, которые высылаются в ответе ведомым устройством ведущему при возникновении какой-либо ошибки [10].

Следует отметить, что на практике чаще всего применяются коды ошибок 0x01, 0x02, 0x03 и 0x04.

Таблица 18

Стандартные коды ошибок протокола ModBus

Код	**Обозначение**	**Краткое описание**
0x01	Illegal function	Принятый код функции не может быть обработан. Это может быть связано с тем, что функциональный код не реализован в выбранном устройстве
0x02	Illegal data address	Адрес данных, указанный в запросе, недоступен
0x03	Illegal data value	Значение, содержащееся в поле данных запроса, является недопустимой величиной
0x04	Server device failure	Произошла невосстанавливаемая ошибка, когда ведомое устройство пыталось выполнить запрошенное действие
0x05	Acknowledge	Ведомое устройство приняло запрос и обрабатывает его, но это требует много времени. Этот ответ предохраняет ведущее устройство от генерации ошибки тайм-аута
0x06	Server device busy	Ведомое устройство занято обработкой команды. Ведущее устройство должно повторить запрос позже, когда ведомое освободится
0x08	Memory parity error	Ведомое устройство при чтении расширенной памяти обнаружило ошибку контроля четности. Ведущее устройство может повторить запрос позже, но обычно в таких случаях требуется ремонт оборудования
0x0A	Gateway path unavailable	Специализированное использование в сочетании со шлюзами указывает на то, что шлюз неправильно настроен или перегружен
0x0B	Gateway target device failed to respond	Специализированное использование в сочетании со шлюзами указывает на отсутствие ответа от ведомого устройства. Обычно означает, что ведомое устройство отсутствует в сети

3.2.2. Controller Area Network

CAN (*англ.* Controller Area Network – сеть контроллеров) – стандарт для построения распределенных промышленных сетей, ориентированный прежде всего на объединение в сеть различных исполнительных устройств и датчиков. Режим передачи последовательный, широковещательный, пакетный, с весьма высокой степенью надежности и защищенности.

CAN разработан компанией Robert Bosch GmbH в 1983 г. первоначально для автомобильной промышленностии, широко распространен в промышленной автоматизации, технологиях домашней автоматизации («умного дома»), автомобильной промышленности и во многих других областях.

CAN охватывает два уровня модели OSI – физический и канальный (см. данные на с. 39). Стандарт не предусматривает никакого протокола прикладного (7-го) уровня модели OSI. Поэтому для его воплощения в жизнь различные фирмы разработали несколько таких протоколов: CANopen (организации CiA), SDS (фирмы «Хонвелл Микро Свитч Дивижн»), CAN Kingdom (фирмы «Квазер»), DeviceNet (фирмы «Аллен-Бредли», ставший европейским стандартом в 2002 г.) и др.

Основные характеристики протокола CAN:

- очень высокая надежность и защищенность,
- каждое сообщение (не устройство) имеет свой собственный приоритет,
- реализован механизм обнаружения ошибок и сигнализация их наличия,
- автоматическая повторная отправка сообщений, которые были доставлены с ошибкой, сразу, как только сеть станет свободной,
- уже упомянутый широковещательный режим передачи данных,
- возможность присутствия нескольких ведущих устройств в одной сети,
- широкий диапазон скоростей приема-передачи,
- высокая устойчивость интерфейса к помехам,
- гибкость конфигурирования и возможность модернизации системы.

К недостаткам стандарта можно отнести сравнительно высокую стоимость CAN-устройств, отсутствие единого протокола прикладного уровня, а также

чрезмерную сложность и запутанность протоколов канального и прикладного уровня, изложенных в стандартах организации CAN in Automation (CiA).

Физический уровень

Физический уровень CAN обеспечивает надежную передачу информации, игнорируя её содержание. Основными понятиями физического уровня являются линии передачи (чаще всего это витая пара, хотя допускается использовать плоский кабель или даже один провод и «корпусную землю»), временные диаграммы, система синхронизации, формат данных, обеспечение достоверности передачи (контрольная сумма, методы кодирования, обнаружение и восстановление ошибок). Характеристики микросхемы приёмопередатчика (транссивера) стандартом не устанавливаются, поскольку они могут быть выбраны для каждого конкретного случая исходя из требований применения. Существует несколько вариантов реализации физических уровней протокола CAN (ISO 11898, ISO 11519, SAE J2411).

В подавляющем большинстве случаев используется физический уровень CAN, который определен в стандарте ISO-11898 [11]. В данном стандарте в качестве среды передачи определяется двухпроводная дифференциальная линия, на концах которой, как и при построении сети ModBus, требуется использовать «терминаторы» 120 Ом (допускается колебание сопротивления в пределах 108–132 Ом). Промышленная сеть CAN представляет собой сеть с общей средой передачи данных. В данном случае все узлы сети одновременно принимают сигналы, передаваемые по CAN-шине, таким образом невозможно послать сообщение какому-либо конкретному узлу. Однако CAN-периферия микроконтроллеров предоставляет аппаратную возможность фильтрации CAN-сообщений.

На рис. 3.10 показана топология сети CAN с подключенными к ней устройствами. Каждое устройство состоит из двух составляющих. Это CAN-транссивер, который обеспечивает взаимодействие сети с микроконтроллером с периферией CAN (CPU), на котором реализован протокол.

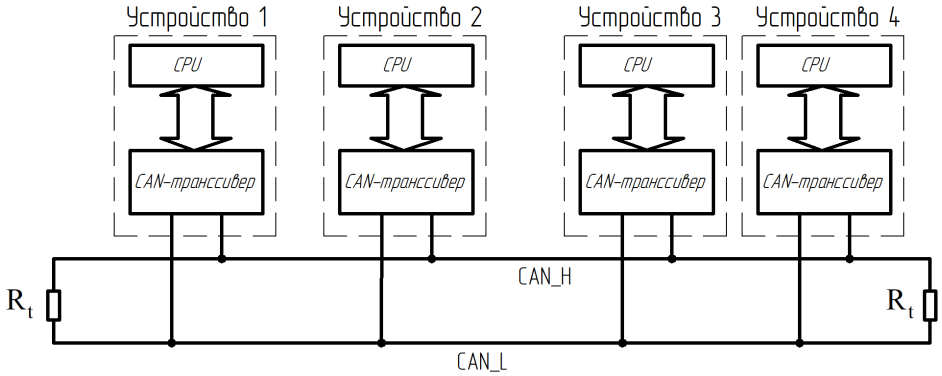


Рис. 3.10. Топология сети CAN

Физический уровень CAN реализован в специальных чипах – CAN-приёмопередатчиках, которые преобразуют обычные TTL-уровни сигналов, используемых CAN-периферией микроконтроллера, в уровни сигналов на шине CAN. В CAN-шине целесообразно применять гальваническую развязку. Для этого производителями микроэлектроники разработаны специальные микросхемы-транссиверы CAN с гальванической развязкой. В качестве примера на рис. 3.11 представлена упрощенная схема внутренних цепей транссивера CAN ISO 1050 DUB с гальванической развязкой.

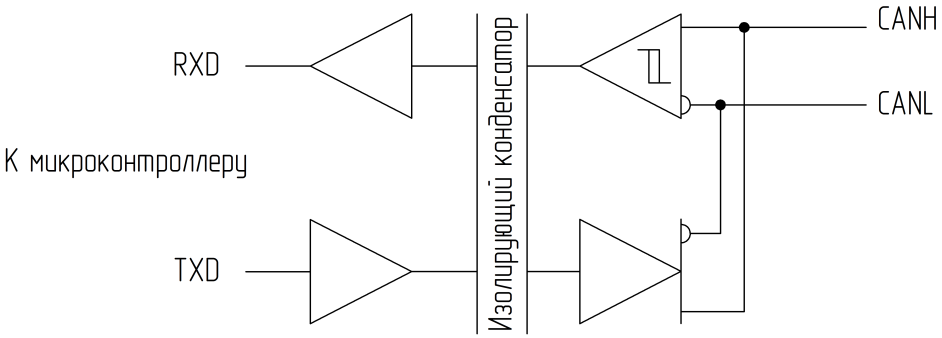


Рис. 3.11. Упрощенная схема внутренних цепей транссивера CAN

CAN-контроллеры соединяются с помощью дифференциальной шины, которая имеет две линии – CANH (CAN-High) и CANL (CAN-Low), по ним передаются сигналы (рис. 3.12). Логический «0» регистрируется, когда на линии CANH сигнал выше, чем на линии CANL. Логическая «1» – когда сигналы CANH и CANL одинаковы (отличаются менее, чем на 0,5 В).

Логический «0» называется доминантным битом, а логическая «1» – рецессивным. Эти названия отражают приоритеты логической единицы и нуля на шине CAN. При одновременной передаче в шину логического нуля и единицы, на шине будет зарегистрирован только логический «0» (доминантный сигнал), а логическая «1» будет подавлена (рецессивный сигнал).

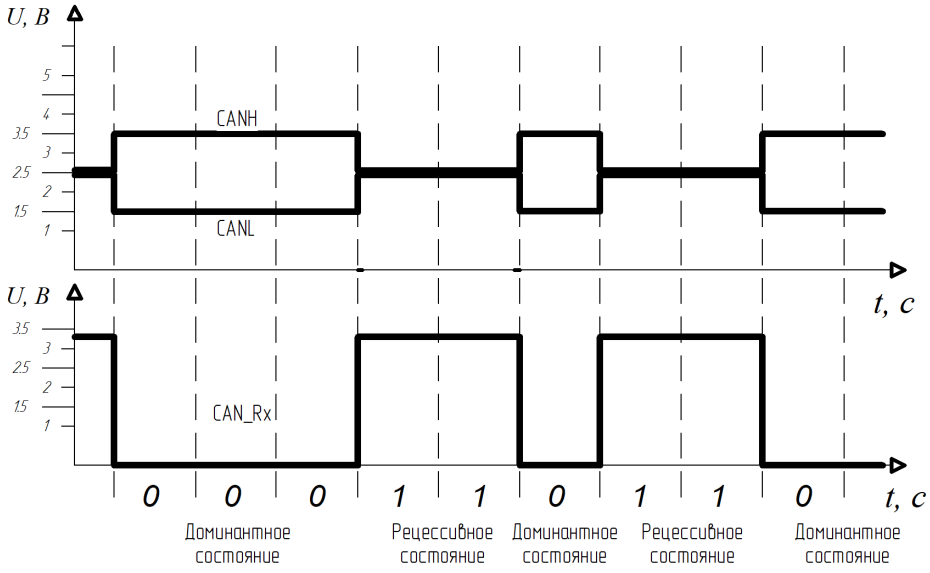


Рис. 3.12. Сигнальные уровни на CAN-шине

Дифференциальная передача данных, как и в ModBus, обеспечивает подавление синфазной помехи, при этом уровень сигналов составляет 1/3 от значения напряжения питания. Следует отметить, что само напряжение питания не определяется жестко. Типичные сигналы с синфазной помехой на CAN-шине при напряжении питания +5 В приведены на рис. 3.13.

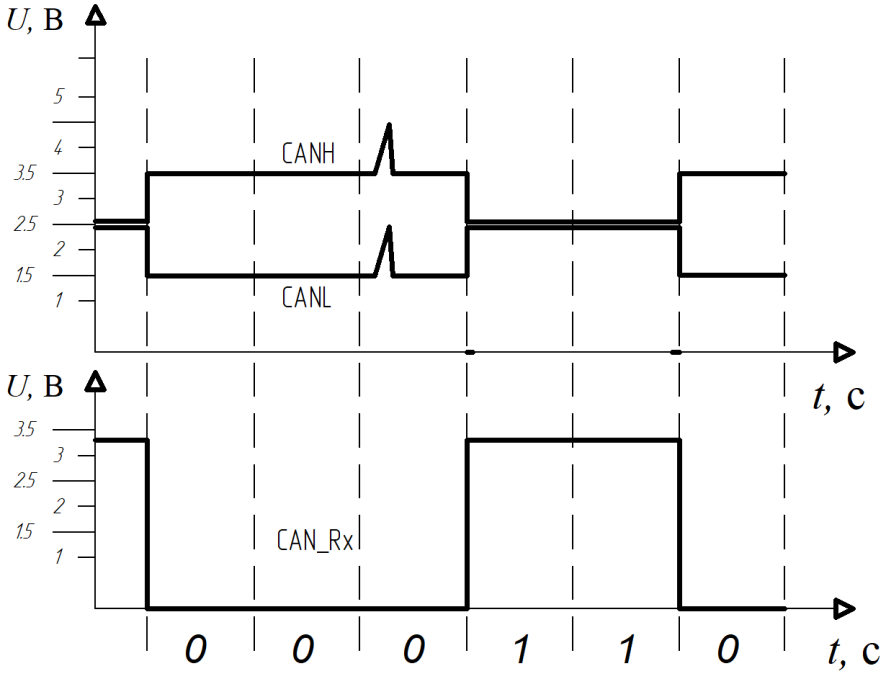


Рис. 3.13. Сигнальные уровни с синфазной помехой на CAN-шине

Максимальная скорость передачи данных по CAN-шине в соответствии с протоколом равна 1 Мбит/с. При такой скорости максимальная длина кабеля равна 40 м. Подобное ограничение на длину кабеля связано с ограничением скорости передачи данных и механизмом побитового арбитража (во время арбитража все узлы сети должны получать текущий бит передачи одновременно, т. е. сигнал должен успеть распространиться по всему кабелю за единичный отсчет времени в сети).

Канальный уровень

Данные в CAN передаются короткими сообщениями-кадрами стандартного формата. В CAN существуют четыре типа сообщений:

– Data Frame,
– Remote Frame,
– Error Frame,
– Overload Frame.

Data Frame – это наиболее часто используемый тип сообщения. Он состоит из следующих основных частей (рис. 3.14).

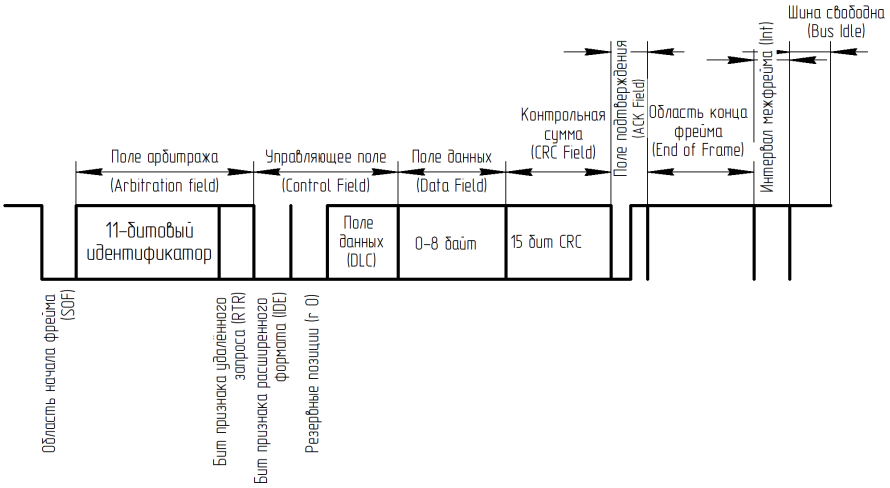


Рис. 3.14. Тип сообщения Data frame стандарта CAN 2.0A

Полем арбитража определяется приоритет сообщения в случае, когда два или более узлов одновременно пытаются передать данные в сеть. Поле арбитража состоит из 11-битного идентификатора + 1 бит RTR (retransmit) (стандарт CAN-2.0A) или из расширенного 29-битного идентификатора + 1 бит RTR (retransmit) (стандарт CAN-2.0B).

Поле арбитража CAN-кадра используется в CAN для разрешения коллизий доступа к шине при помощи арбитража, суть которого заключается в следующем (рис. 3.15). Когда несколько контроллеров начинают одновременную передачу CAN-кадра в сеть, каждый из них сравнивает бит, который собирается передать на шину, с битом, который пытается передать на шину конкурирующий контроллер. Если значения этих битов равны, оба контроллера передают следующий бит. И так происходит до тех пор, пока значения передаваемых битов не окажутся различными. Теперь контроллер, который передавал логический «0» (более приоритетный сигнал), будет продолжать передачу, а другой (другие) контроллер прервёт свою передачу до того времени, пока шина вновь не освободится. Конечно, если шина в данный момент занята, то контроллер не начнет передачу до момента её освобождения.

Следует отметить, что поле идентификатора, несмотря на свое название, не идентифицирует узел в сети или содержимое поля данных. В CAN-сети узлы не имеют адреса. Таким образом сообщение принимается узлом, если его идентификатор проходит через фильтр сообщений, который запрограммирован в каждом узле. Для Data-кадра бит RTR всегда выставлен в логический «0» (доминантный сигнал).

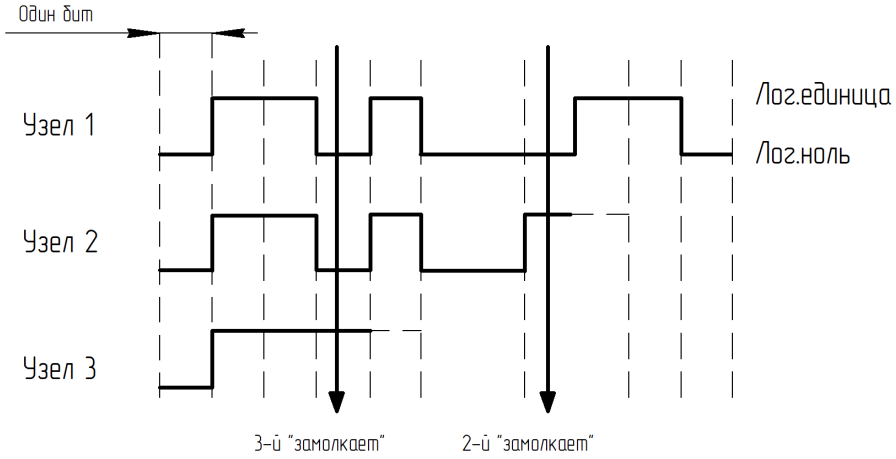


Рис. 3.15. Побитовый арбитраж на шине CAN

Управляющее поле Control Field, показанное на рис. 3.16, содержит 6 бит, из которых 4 бита (DLC0–DLC4) составляют поле Data Length Code, показывающее количество байтов данных, которое будет передаваться в поле данных. Два других бита зарезервированы для следующих редакций протокола.

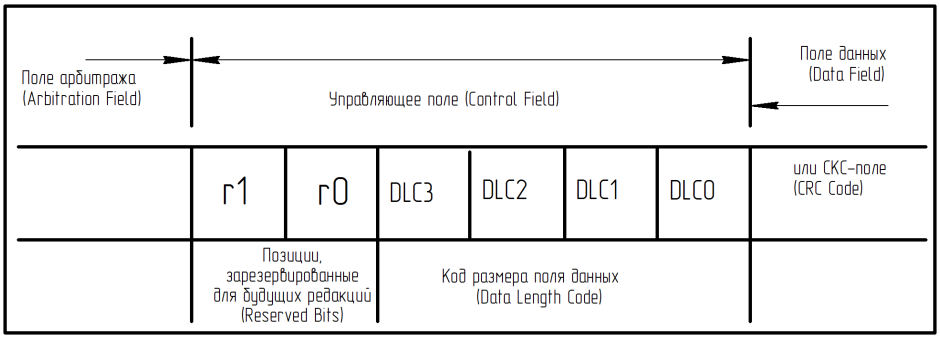


Рис. 3.16. Управляющее поле фрейма данных

Поле данных (data field) содержит 0–8 байт данных. Поле CRC (CRC field) содержит 15-битную контрольную сумму сообщения, которая используется для обнаружения ошибок. Поле подтверждения ACK Field, показанное на рис. 3.17, содержит участки ACK Slot и ACK Delimiter и выполняет следующую функцию: передающий узел посылает по одному рецессивному биту на каждом из участков, а приемник, если он принял сообщение без сбоев, устанавливает на линии доминирующий бит в поле ACK Slot. При наложении рецессивного и доминирующего уровней на линии устанавливается доминирующий, и это событие сигнализирует передающему узлу о том, что передача прошла нормально и повтор не требуется.

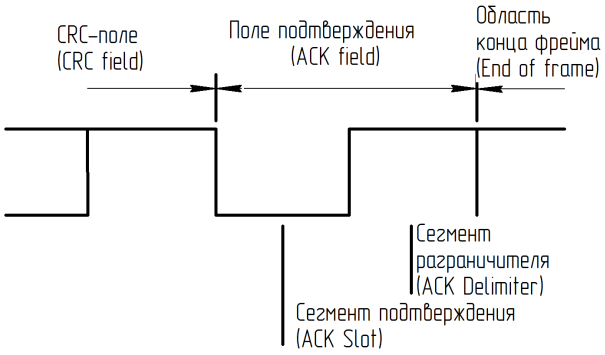


Рис. 3.17. Поле подтверждения фрейма данных

Remote Frame – это часть Data Frame, но без поля данных. В этом поле также выставлен бит RTR (1 – рецессивный бит). Основная задача данного поля – это инициация одним из узлов сети передачи данных другим узлом. Подобная схема позволяет уменьшить общий трафик CAN-шины. Следует отметить, что на практике Remote Frame используется редко.

Error Frame – это сообщение, которое явно нарушает формат сообщения CAN. Передача данного сообщения приводит к тому, что все узлы сети регистрируют ошибку формата передаваемого кадра и автоматически передают в сеть Error Frame. В итоге осуществляется автоматическая повторная передача данных в сеть передающим узлом. Error Frame содержит в своём составе поле Error Flag, состоящее из 6 бит одинакового значения, и поля Error Delimiter, состоящее из 8 рецессивных битов. Error Delimiter дает возможность другим узлам сети, обнаружив Error Frame, послать в сеть свой Error Flag.

Overload Frame повторяет структуру и логику работы кадра Error Frame, но в данном случае он используется перегруженным узлом, который не может обработать поступающее сообщение и поэтому просит при помощи Overload-кадра о повторной передаче данных. Следует отметить, что Overload-кадр практически не используется.

CAN-протокол определяет пять способов обнаружения ошибок в сети:

– Bit monitoring,
– Bit stuffing,
– Frame check,
– ACKnowledgement Check,
– CRC Check.

Bit monitoring – каждый узел во время передачи данных в CAN-сеть сравнивает значение передаваемого им бита со значением бита, которое появляется на шине. Если отправленные и читаемые значения не совпадают, то узел генерирует ошибку Bit Error. Во время арбитража на CAN-шине этот механизм проверки ошибок отключается.

Bit stuffing – в тот момент, когда узел передает последовательно в шину 5 бит с одинаковым значением, он добавляет шестой бит с противоположным значением. Принимающие узлы этот дополнительный бит удаляют самостоятельно. Если узел обнаруживает на шине больше 5 последовательных бит с одинаковым значением, то он генерирует ошибку Stuff Error.

Frame Check – некоторые части CAN-сообщения имеют одинаковое значение во всех типах сообщений. В данном случае протокол CAN сам определяет, какие уровни напряжения и когда должны появляться на шине. Если формат сообщений нарушается, то узлы генерируют ошибку Form Error.

ACKnowledgement Check – каждый узел, получив правильное сообщение по сети, посылает в сеть доминантный («0») бит. Если подобного не происходит, передающий узел регистрирует ошибку Acknowledgement Error.

CRC Check – каждое сообщение CAN содержит CRC; подобно сообщению ModBus RTU, каждый принимающий узел вычисляет значение CRC для каждого полученного сообщения. Если подсчитанное значение CRC не совпадает со значением CRC в принимаемом сообщении, принимающий узел генерирует ошибку CRC Error.

Устройство, обнаружившее любую из перечисленных ошибок, сигнализирует об этом с помощью флага ошибки.

Прикладной уровень

Как было отмечено выше, в CAN реализовано только первых два уровня модели OSI, поэтому существует ряд несовместимых между собой протоколов прикладного уровня. Самыми распространенными из них являются CANopen и DeviceNet.

CANopen разработан как стандартизированная встроенная сеть с гибкими возможностями настройки [12]. Первоначально он был предложен для систем управления машинами, ориентированных на перемещение, таких как системы для погрузочно-разгрузочных работ. Сегодня он используется в различных областях: медицинское оборудование, внедорожники, морская электроника, железнодорожное оборудование, а также автоматизация зданий и сооружений.

CANopen построен на верхнем уровне CAL (CAN Application Layer) и представляет только часть функций CAL, оптимизированных для упрощённых систем, которые работают в режиме реального времени.

Центральное понятие в CANopen – это объектный словарь устройства (Device Object Dictionary). Словарь упорядочен по группам объектов; каждый объект адресуется с использованием 16-битного индекса. Для доступа к отдельным элементам в структурах используется 8-битный субиндекс.

Объектный словарь CANopen:

Индекс	**Объект**
0x0000	Зарезервировано
0x0001…0x025F	Типы данных
0x0260…0x0FFF	Зарезервировано
0x1000…0x1FFF	Область профиля обмена данными (тип устройства, регистр ошибок)
0x2000…0x5FFF	Область профиля, заданного производителем
0x6000…0x9FFF	Стандартизированный профиль устройства
0xA000…0xBFFF	Область профиля интерфейса
0xC000…0xFFFF	Зарезервировано

Все параметры связи и приложений, определяющие поведение устройства CANopen, администрируются в объектном словаре CANopen. Любая запись в словаре объектов CANopen доступна для чтения через службы связи CANopen. Пример использования CAN приведен в пп. 3.3.2 «Программирование CAN на языке С».

3.3. Примеры программирования

3.3.1. Программирование ModBus на языке С

Для использования функций ModBus на микроконтроллерах любых производителей необходимо спараметрировать аппаратную часть микроконтроллера – модуль USART (Universal synchronous and asynchronous receiver-transmitter), к которому подключается микросхема – приемопередатчик, показанная на рис. 3.5. При параметрировании микроконтроллера необходимо указать скорость передачи данных, паритет и др. Пример параметрирования модуля USART для микроконтроллера Stm32 показан далее.

```
1  static void MX_USART1_UART_Init(void)
2  {
3      huart1.Instance = USART1;
4      huart1.Init.BaudRate = 19200;
5      huart1.Init.WordLength = UART_WORDLENGTH_9B;
6      huart1.Init.StopBits = UART_STOPBITS_1;
7      huart1.Init.Parity = UART_PARITY_EVEN;
8      huart1.Init.Mode = UART_MODE_TX_RX;
9      huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
10     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
11     huart1.Init.OneBitSampling =
   ↪   UART_ONE_BIT_SAMPLE_DISABLE;
12     huart1.AdvancedInit.AdvFeatureInit =
   ↪   UART_ADVFEATURE_NO_INIT;
```

```
13      if (HAL_RS485Ex_Init(&huart1, UART_DE_POLARITY_HIGH,
    ↪   0, 0) != HAL_OK)
14      {
15          Error_Handler();
16      }
17  }
```

Параметрирование модуля USART выполняется одинаково для функций ModBus Slave и ModBus Master.

ModBus Slave

Первым шагом в реализации ModBus Slave на ПЛК определяется функция, которую будет выполнять программа, в зависимости от кода функции, который посылается ведущим устройством. Для этого проверяется значение второго байта и выполняется соответствующая функция.

В данном примере приведены функции чтения состояния дискретных выходов ПЛК, чтения состояния дискретных входов ПЛК, чтения данных их регистров хранения, запись состояния одного дискретного выхода и функция записи одного регистра хранения данных.

```
1   void ModBusRTU_Slave_readData(void) {
2       if (RxData[0] == SLAVE_ID) {
3           switch (RxData[1]) {
4               case 0x01: readCoils(); break;
5               case 0x02: readInputs(); break;
6               case 0x03: readHoldingRegs(); break;
7               case 0x05: writeSingleCoil(); break;
8               case 0x06: writeSingleReg(); break;
9               default: modbusException(ILLEGAL_FUNCTION);
    ↪   break;
10          }
11      }
12  }
```

Пример функции, исполняющей отработку кода функции 0x01, показан ниже. В нём осуществляется проверка количества и диапазона считываемых данных. Если количество запрашиваемых дискретных выходов превышает доступное и запрашиваемый адрес выходит за пределы адреса в памяти ведомого устройства, то выполняется функция *modbusException*, которая отсылает ведущему устройству сообщение о соответствующей ошибке. Далее побайтово формируется ответ, который содержит в своём составе адрес ведомого устройства, код функции, количество байтов данных и состояние дискретных выходов, которые необходимо передать ведущему устройству. При отправке данных, в функции *sendData* рассчитывается CRC и добавляется к отсылаемому кадру.

Данная функция используется в целях чтения состояний дискреных выходов ПЛК для выставления того или иного управляющего воздействия, для демонстрации состояния дискретного выхода в графическом интерфейсе SCADA-системы при рассмотрении примера (гл. 4) комплексной системы автоматизации зданий и сооружений.

```
uint8_t readCoils (void) {
    uint16_t startAddr = ((RxData[2]<<8)|RxData[3]);
    uint16_t numCoils = ((RxData[4]<<8)|RxData[5]);
    if ((numCoils<1)||(numCoils>2000)) {
        modbusException (ILLEGAL_DATA_VALUE);
        return 0;
    }
    uint16_t endAddr = startAddr+numCoils-1;
    if (endAddr>199) {
        modbusException(ILLEGAL_DATA_ADDRESS);
        return 0;
    }
    memset (TxData, '\0', 256);
    TxData[0] = SLAVE_ID;
    TxData[1] = RxData[1];
    TxData[2] = (numCoils/8) + ((numCoils%8)>0 ? 1:0);
    int indx = 3;
    int startByte = startAddr/8;
```

```
19      uint16_t bitPosition = startAddr%8;
20      int indxPosition = 0;
21      for (int i=0; i<numCoils; i++) {
22          TxData[indx] |= ((Coils_Database[startByte] >>
    ↪  bitPosition) &0x01) << indxPosition;
23          indxPosition++; bitPosition++;
24          if (indxPosition>7) {
25              indxPosition = 0;
26              indx++;
27          }
28          if (bitPosition>7) {
29              bitPosition=0;
30              startByte++;
31          }
32      }
33      if (numCoils%8 != 0)indx++;
34      sendData(TxData, indx);
35      return 1;
36  }
```

Пример функции, исполняющей отработку кода функции 0x02, показан далее. В нём осуществляется проверка количества и диапазона считываемых данных. Если количество запрашиваемых дискретных входов превышает доступное и запрашиваемый адрес выходит за пределы адреса в памяти ведомого устройства, то выполняется функция *modbusException*, которая отсылает ведущему устройству сообщение о соответствующей ошибке. Далее побайтово формируется ответ, который содержит в своём составе адрес ведомого устройства, код функции, количество байтов данных и состояние дискретных входов, которые необходимо передать ведущему устройству. При отправке данных, в функции *sendData* рассчитывается CRC и добавляется к отсылаемому кадру.

Функция чтения состояния дискретных входов может активно применяться как при работе ПЛК в составе распределённой системы автоматики, так и для демонстрации состояния дискретных входов в графическом интерфейсе SCADA-системы при построении комплексной системы автоматизации зданий

и сооружений. При использовании данной функции оператор видит, что на дискретный вход ПЛК пришёл сигнал, соответственно либо оператор, либо система автоматики выполняет действие, заложенное в алгоритме всей системы.

```
uint8_t readInputs (void) {
    uint16_t startAddr = ((RxData[2]<<8)|RxData[3]);
    uint16_t numInputs = ((RxData[4]<<8)|RxData[5]);
    if ((numInputs<1)||(numInputs>2000)) {
        modbusException (ILLEGAL_DATA_VALUE);
        return 0;
    }
    uint16_t endAddr = startAddr+numInputs-1;
    if (endAddr>199) {
        modbusException(ILLEGAL_DATA_ADDRESS);
        return 0;
    }
    memset (TxData, '\0', 256);
    TxData[0] = SLAVE_ID;
    TxData[1] = RxData[1];
    TxData[2] = (numInputs/8) + ((numInputs%8)>0 ? 1:0);
    int indx = 3;
    int startByte = startAddr/8;
    uint16_t bitPosition = startAddr%8;
    int indxPosition = 0;
    for (int i=0; i<numInputs; i++) {
        TxData[indx] |= ((Inputs_Database[startByte] >>
↪ bitPosition) &0x01) << indxPosition;
        indxPosition++; bitPosition++;
        if (indxPosition>7) {
            indxPosition = 0;
            indx++;
        }
        if (bitPosition>7) {
            bitPosition=0;
```

```
30                 startByte++;
31             }
32         }
33         if (numInputs%8 != 0)indx++;
34         sendData(TxData, indx);
35         return 1;
36     }
```

Функция, исполняющая отработку кода функции 0x03, аналогична предыдущим и показана ниже. В данном примере осуществляется проверка количества и диапазона считываемых данных. Если количество запрашиваемых регистров хранения данных превышает доступное и запрашиваемый адрес выходит за пределы адреса в памяти ведомого устройства, то выполняется функция *modbusException*, которая отсылает ведущему устройству сообщение о соответствующей ошибке. Далее побайтово формируется ответ, который содержит в своём составе адрес ведомого устройства, код функции, количество байтов данных и данные, хранящиеся в запрашиваемых регистрах, которые необходимо передать ведущему устройству. При отправке данных, в функции *sendData* рассчитывается CRC и добавляется к отсылаемому кадру.

Функция чтения содержимого регистров хранения данных может активно применяться как при работе ПЛК в составе распределённой системы автоматики, передавая данные для управления различными объектами, например насосными станциями, так и для демонстрации значения необходимой переменной, находящейся в регистре хранения в графическом интерфейсе SCADA-системы при построении комплексной системы автоматизации зданий и сооружений. При использовании данной функции оператор видит значение переменной, хранящейся в запрашиваемом регистре, соответственно либо оператор, либо система автоматики выполняет действие, заложенное в алгоритме всей системы.

```
1  uint8_t readHoldingRegs (void) {
2      uint16_t startAddr = ((RxData[2]<<8)|RxData[3]);
3      uint16_t numRegs = ((RxData[4]<<8)|RxData[5]);
4      if ((numRegs<1)||(numRegs>125)) {
5          modbusException (ILLEGAL_DATA_VALUE);
```

```
6           return 0;
7       }
8       uint16_t endAddr = startAddr+numRegs-1;
9       if (endAddr>49) {
10          modbusException(ILLEGAL_DATA_ADDRESS);
11          return 0;
12      }
13      TxData[0] = SLAVE_ID;
14      TxData[1] = RxData[1];
15      TxData[2] = numRegs*2;
16      int indx = 3;
17      for (int i=0; i<numRegs; i++) {
18          TxData[indx++] =
    ↪  (Holding_Registers_Database[startAddr]>>8)&0xFF;
19          TxData[indx++] =
    ↪  (Holding_Registers_Database[startAddr])&0xFF;
20          startAddr++;
21      }
22      sendData(TxData, indx);
23      return 1;
24  }
```

Следующая функция *writeSingleCoil* исполняет отработку кода функции 0x05. В данном примере осуществляется проверка адреса записываемых данных. Если запрашиваемый адрес выходит за пределы адреса в памяти ведомого устройства, то выполняется функция *modbusException*, которая отсылает ведущему устройству сообщение о соответствующей ошибке. Далее побайтово формируется ответ, который содержит в своём составе адрес ведомого устройства, код функции, адрес дискретного выхода и значение, которое необходимо установить на соответствующем дискретном выходе. При отправке данных, в функции *sendData* рассчитывается CRC и добавляется к отсылаемому кадру.

Данная функция записи состояния одного дискретного выхода может активно применяться при работе ПЛК в составе распределённой системы автоматики, для управления различными объектами, например включением

освещения здания. При использовании данной функции система автоматики выполняет действие, заложенное в алгоритме системы, подавая на нужный дискретный выход ПЛК необходимое значение ВКЛ или ВЫКЛ.

```
uint8_t writeSingleCoil (void) {
    uint16_t startAddr = ((RxData[2]<<8)|RxData[3]);
    if (startAddr>NumberOfCoils-1) {
        modbusException(ILLEGAL_DATA_ADDRESS);
        return 0;
    }
    int startByte = startAddr/8;
    uint16_t bitPosition = startAddr%8;
    if ((RxData[4] == 0xFF) && (RxData[5] == 0x00)) {
        Coils_Database[startByte] |= 1<<bitPosition;
    }
    else if ((RxData[4] == 0x00) && (RxData[5] == 0x00))
    {
        Coils_Database[startByte] &= ~(1<<bitPosition);
    }
    TxData[0] = SLAVE_ID;
    TxData[1] = RxData[1];
    TxData[2] = RxData[2];
    TxData[3] = RxData[3];
    TxData[4] = RxData[4];
    TxData[5] = RxData[5];
    sendData(TxData, 6);
    return 1;
}
```

Функция *writeSingleReg(void)*, исполняет отработку кода функции 0x06. В данном примере осуществляется проверка адреса записываемого регистра. Если запрашиваемый адрес выходит за пределы адреса в памяти ведомого устройства, то выполняется функция *modbusException*, которая отсылает ведущему устройству сообщение о соответствующей ошибке. Далее побайтово фор-

мируется ответ, который содержит в своём составе адрес ведомого устройства, код функции, адрес регистра и значение, которое необходимо установить в соответствующем регистре хранения данных. При отправке данных, в функции *sendData* рассчитывается CRC и добавляется к отсылаемому кадру.

Функция записи одного регистра хранения данных может активно применяться при работе ПЛК в составе распределённой системы автоматики, для управления различными объектами, например установкой задания на скорость вращения насоса в системе водоснабжения здания. При использовании данной функции система автоматики выполняет действие, заложенное в алгоритме системы, записывая в нужный регистр хранения данные, которые впоследствии будут переданы далее на обработку.

```
uint8_t writeSingleReg (void) {
    uint16_t startAddr = ((RxData[2]<<8)|RxData[3]);
    if (startAddr>49) {
        modbusException(ILLEGAL_DATA_ADDRESS);
        return 0;
    }
    Holding_Registers_Database[startAddr] =
    ↪ (RxData[4]<<8)|RxData[5];
    TxData[0] = SLAVE_ID;
    TxData[1] = RxData[1];
    TxData[2] = RxData[2];
    TxData[3] = RxData[3];
    TxData[4] = RxData[4];
    TxData[5] = RxData[5];
    sendData(TxData, 6);
    return 1;
}
```

ModBus Master

Функцию ModBus Master можно оформить в виде функции, аргументами которой являются код функции и данные, которые необходимо передать. В зави-

симости от кода функции проводится построение кадра. Ниже приведён пример построения кадра для кода функции 0x06 в целях записи одного регистра хранения данных в удаленном устройстве. В данной функции побайтово выстраивается кадр, к нему добавляется вычисленная контрольная сумма и далее всё передается при помощи функции HAL_UART_Transmit_IT в удаленное устройство.

```
1  void Modbus(uint16_t Mdb_Reg, uint16_t Mdb_Data)
2  {
3      Mdb_TxData[0] = VFD_MODBUS_Addres;
4      Mdb_TxData[1] = Write_One_Output_Word;
5      Mdb_TxData[2] = (Mdb_Reg >> 8) & 0xFF;
6      Mdb_TxData[3] = Mdb_Reg & 0xFF;
7      Mdb_TxData[4] = (Mdb_Data >> 8) & 0xFF;
8      Mdb_TxData[5] = Mdb_Data & 0xFF;
9      commsBuffer[0] = Mdb_TxData[0];
10     commsBuffer[1] = Mdb_TxData[1];
11     commsBuffer[2] = Mdb_TxData[2];
12     commsBuffer[3] = Mdb_TxData[3];
13     commsBuffer[4] = Mdb_TxData[4];
14     commsBuffer[5] = Mdb_TxData[5];
15     CRC16 = HAL_CRC_Calculate(&hcrc,
   ↪   (uint32_t*)&commsBuffer, 0x06);
16     Mdb_TxData[6] = CRC16 & 0xFF;
17     Mdb_TxData[7] = (CRC16 >> 8) & 0xFF;
18     HAL_UART_Transmit_IT(&huart1, (uint8_t *)Mdb_TxData,
   ↪   sizeof(Mdb_TxData));
19 }
```

3.3.2. Программирование CAN на языке C

Пример параметрирования аппаратного блока микроконтроллера Stm32 (модуля CAN) показан ниже. При параметрировании микроконтроллера указывается скорость передачи данных, тип идентификаторов, с которыми будет работать модуль CAN, и т. д.

```
static void MX_CAN_Init(void) {
    hcan.Instance = CAN;
    hcan.Init.Prescaler = 4;
    hcan.Init.Mode = CAN_MODE_NORMAL;
    hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
    hcan.Init.TimeSeg1 = CAN_BS1_13TQ;
    hcan.Init.TimeSeg2 = CAN_BS2_2TQ;
    hcan.Init.TimeTriggeredMode = DISABLE;
    hcan.Init.AutoBusOff = ENABLE;
    hcan.Init.AutoWakeUp = DISABLE;
    hcan.Init.AutoRetransmission = ENABLE;
    hcan.Init.ReceiveFifoLocked = DISABLE;
    hcan.Init.TransmitFifoPriority = ENABLE;
    if (HAL_CAN_Init(&hcan) != HAL_OK) {
        Error_Handler();
    }
    TxHeader.StdId = 0x0378;
    TxHeader.ExtId = 0;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.DLC = 8;
    TxHeader.TransmitGlobalTime = 0;
    HAL_CAN_Start(&hcan);
    HAL_CAN_ActivateNotification(&hcan,
↪   CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_ERROR |
↪   CAN_IT_BUSOFF | CAN_IT_LAST_ERROR_CODE);
    CAN_FilterTypeDef  sFilterConfig;
    sFilterConfig.FilterBank = 0;
    sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
    sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
    sFilterConfig.FilterIdHigh = 0x0000;
    sFilterConfig.FilterIdLow = 0x0000;
    sFilterConfig.FilterMaskIdHigh = 0x0000;
```

```
32      sFilterConfig.FilterMaskIdLow = 0x0000;
33      sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
34      sFilterConfig.FilterActivation = ENABLE;
35      if(HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) !=
    ↪   HAL_OK) {
36          Error_Handler();
37      }
38  }
```

Приём данных осуществляется в прерывании. Если данные пришли без ошибок, то проверяется идентификатор, и если идентификатор принятого сообщения совпадает с идентификатором, указанным в программе в объектном словаре, то начинается выполнение пользовательской программы. Пример приёма данных по CAN-шине с последующим выполнением пользовательской программы показан далее.

```
1   void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef
    ↪   *hcan)
2   {
3       if(HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0,
    ↪   &RxHeader, CAN_RxData) == HAL_OK)
4       {
5           if(RxHeader.StdId == USER_CANID)
6           {
7               //Пользовательская программа
8           }
9       }
10  }
```

Пример передачи данных по CAN-шине показан далее. В нем передается пользовательский массив данных CAN_TxData с идентификатором CAN_ID. Даннный участок кода можно выделить в отдельную функцию, в которую в качестве аргуменов можно отправлять данные для передачи по CAN-шине.

```
TxHeader.StdId = CAN_ID;
while(HAL_CAN_GetTxMailboxesFreeLevel(&hcan) == 0);
if(HAL_CAN_AddTxMessage(&hcan, &TxHeader, CAN_TxData,
 ↪  &TxMailbox) != HAL_OK)
{
}
```

Конрольные темы для повторения

1. Сетевая модель OSI. Описание и функции уровней.
2. Физический уровень ModBus RTU.
3. Канальный уровень ModBus RTU.
4. Прикладной уровень ModBus RTU.
5. Физический уровень CAN.
6. Канальный уровень CAN.
7. Пример прикладного уровня CAN.

Глава 4. Пример применения ПЛК

4.1. Описание учебного стенда

Назначение учебно-демонстрационного комплекса приёма-передачи данных[1] – проведение лабораторных работ по настройке систем связи для различных задач автоматизации, параметрирование преобразователя частоты и написание собственной программы для контроллера (рис. 4.1).

Рис. 4.1. Внешний вид учебного стенда

[1] В разработке стенда принимал активное участие магистрант каф. ЭАПУ УралЭНИН УрФУ А.С. Кудряшов.

На стенде реализованы следующие системы связи:

– ModBus RTU посредством преобразователя USB/RS-485,
– ModBus TCP/IP посредством Modbus-шлюза MOXA MGate MB3180,
– IoT технологии посредством сетевого шлюза ОВЕН ПВ210.

Учебный стенд позволяет выполнить следующие задачи:

– удаленный сбор данных учета электрической энергии и расхода воды,
– удаленное управление исполнительными механизмами,
– удаленный контроль аварийных ситуаций (посредством датчика протечки воды).

4.2. Оборудование и подготовка учебного стенда к работе

4.2.1. Оборудование

В этом разделе рассмотрено основное оборудование, которым удалённо управляют и которое передаёт данные, одним словом, диспетчеризируется. На практике, для общего понимания состава оборудования и его работы, используют структурные схемы. Структурная схема учебного стенда показана на рис. 4.2.

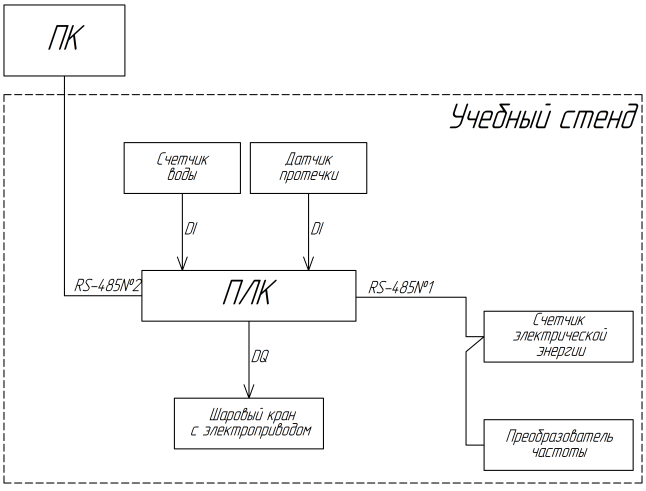


Рис. 4.2. Структурная схема учебного стенда

Программируемый логический контроллер (рис. 4.3). Контроллер разработан и произведен в России, на кафедре «Электропривод и автоматизация промышленных установок» УралЭНИН УрФУ. В нем задается логика опроса датчиков и управления исполнительными устройствами стенда. При помощи ПЛК, кроме того, настраивается связь с исполнительными устройствами на стенде и ПК для удаленного управления и сбора данных. Технические характеристики ПЛК представлены ниже.

Рис. 4.3. Внешний вид ПЛК

Технические характеристики ПЛК:

Источник питания	
Входное напряжение 1	24 В DC
Допустимый диапазон 1	12,0–28,8 В DC
Входное напряжение 2	5 В DC
Допустимый диапазон 2	4–20 В DC
Цифровые входы	
Количество	8
Гальваническое разделение цепей	Есть
Аналоговые входы	Нет
Цифровые выходы	
Количество	4
Тип выхода	Транзисторные, PNP
Выходной ток	До 0,3 А на канал
Сетевые шины	
RS-485	2

Универсальный крыльчатый одноструйный сухоходный счетчик воды ПУЛЬС-15УИ-80 для холодной и горячей воды (рис. 4.4). Счётчик предназначен для измерения объемного расхода холодной питьевой воды по СанПиН 2.1.4.1074–01 и сетевой воды, протекающей по трубопроводам систем горячего (ГВС) и холодного (ХВС) водоснабжения при давлении до 1,6 МПа и в диапазоне температур +5…+90 °С. Классы точности – А и В.

Рис. 4.4. Счетчик воды

Счетчик оснащен импульсным выходом, который основан на воздействии магнитного поля постоянного магнита на геркон, при этом контакты геркона замыкаются и размыкаются. Геркон формирует пассивный выходной сигнал в виде «сухого контакта» каждый раз при прохождении 10 л воды через крыльчатку счетчика. Этот сигнал подается на дискретный вход контроллера.

Датчик протечки воды GIDROLOCK WSP (рис. 4.5). Датчик предназначен для обнаружения протечек воды в системах водоснабжения, отопления, канализации и кондиционирования.

Технические характеристики: длина кабеля – 3 м; ток потребления датчика WSP – 0 мА; температурный диапазон эксплуатации – -30 …+ 60°С; размеры (без кабеля) (Ш/В/Д) – 35 х 8 х 47 мм.

При попадании воды на его электроды, электрическая цепь замыкается, передавая сигнал на дискретный вход контроллера.

Рис. 4.5. Датчик протечки воды

Шаровый кран с электроприводом (рис. 4.6). Кран предназначен для перекрытия воды в системах водоснабжения и отопления. Кран может применяться совместно с контроллером обнаружения протечек воды и как отдельное автономное устройство.

Рис. 4.6. Шаровый кран с электроприводом

Технические характеристики:

– рабочая среда – жидкости, газы;
– напряжение – 220–230 В AC;
– рабочий ток – ≤500 мА;
– рабочий ресурс – 70 000 ч;
– время открытия-закрытия – 15 с;
– угол поворота привода – 90°;
– максимальная сила крутящего момента – 2 Н·м;
– класс защиты – IP50.

Привод крана подключен к контактам реле, катушка которого управляется от дискретного выхода контроллера.

Однофазный многотарифный электросчетчик серии «СЕ» компании «Энергомера» (рис. 4.7). Выполняет измерение активной электроэнергии в однофазных цепях переменного тока; обеспечивает организацию учета по четырем тарифам плюс один (аварийный) тариф с передачей накопленной информации через интерфейсы: оптический, RS-485. Счетчик измеряет и отображает на дисплее параметры сети.

Рис. 4.7. Внешний вид электросчетчика

Основные характеристики представлены ниже.

Нормативно-правовое обеспечение соответствует ГОСТ 31819.21–2012, ГОСТ 31818.11–2012.

Функциональные возможности:

– ведение архивов потребленной активной энергии по 4 + 1 тарифам и суммарно – нарастающим итогом (всего от обнуления); за текущий и 12 предыдущих месяцев; за текущие и 36 предыдущих суток; на конец суток, за 36 сут;
– ведение журналов состояния или событий, журналов программирования параметров. В журналах фиксируются события и время, дата их наступления;
– устойчивость ЖК-индикатора к воздействию электромагнитных полей и сохранение своей работоспособности в широком диапазоне температур;
– измерение и отображение параметров сети (ток, напряжение, частота);

– энергонезависимая память;
– повышенная устойчивость к климатическим, механическим, тепловым и электромагнитным воздействиям;
– наличие интерфейсов связи – оптопорт, RS-485 – в зависимости от модификации;
– защита от несанкционированного доступа паролем.

Частотный преобразователь Chifon FPR500A-0.75G-S2 (рис. 4.8). Может использоваться для управления скоростью асинхронных электродвигателей мощностью до 0,75 кВ.

Рис. 4.8. Внешний вид преобразователя частоты

Основные технические характеристики Chifon FPR500A-0.75G-S2:

– номинальное входное напряжение – однофазное AC, 220 В 50/60 Гц;
– номинальная частота входного напряжения – 50 / 60 Гц;
– номинальный ток – 2,5 А;
– номинальная мощность – 0,75 кВт (1 л.с.);
– класс защиты – IP20 (защита от пыли, но не защита от воды);
– температурный режим – −10…+40 °C;
– регулирование частоты – 0–400 Гц.

Встроенные функции: управление скоростью, контроль над направлением вращения, защита от короткого замыкания, защита от перегрузки, автоматический поиск частоты, автоматическое восстановление после сбоя питания.

Преобразователем можно управлять при помощи кнопок на передней панели, а также через интерфейс RS-485.

ПК – персональный компьютер. Можно использовать любое устройство с возможностью подключения USB-преобразователя.

4.2.2. Цифровые интерфейсы

В контроллере предусмотрено две шины RS-485, которые обозначены как RS-485 № 1 и RS-485 № 2 (см. рис. 4.2).

На первой шине RS-485 контроллер является мастером (Master), а счетчик электрической энергии и преобразователь частоты – подчинёнными устройствами (Slave). В этой шине используется два разных протокола передачи данных (CE и ModBus RTU). К шине можно подключить дополнительные устройства через клеммы (XT_RS1), которые предусмотрены на стенде. В программе контроллера для этой шины прописаны параметры.

Параметры шины RS-485 № 1:

Скорость (Baud Rate), бит/с	9600
Бит данных (Data Bits)	8
Стоповые биты (Stop Bits)	1
Проверка четности (Parity)	Чёт. (Even)
Протокол	RTU

Параметры, указанные выше, выбраны исходя из того, что в настройках счетчика электрической энергии нельзя менять значения параметров сети для передачи данных.

Рассмотрим протокол CE, который используется в счетчиках электрической энергии «Энергомера».

Обмен со счетчиком построен по схеме ведущий – ведомый. Счетчик всегда является ведомым, т. е. передает информацию в канал только по запросу ведущего (рис. 4.9).

Поле	END	OPT	AddrD		AddrS		Passw				Serv	AddrH	AddrL	Data		CRC8	END
№ байта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Значение	C0	48	69	91	FD	00	31	DE	0B	00	D2	01	30	00	00	E0	C0

Рис. 4.9. Структура сообщения при запросе протокола CE

END – флаг, обозначающий начало и конец пакета, всегда равен 0xC0.

OPT равен 0x48, т. е. включена 16-битная адресация и 8-битный циклический код.

AddrD – адрес назначения (счетчика), адрес 2-байтовый (младший байт передается первым).

AddrS – адрес источника, адрес 2-байтовый (младший байт передается первым).

CRC8 – контрольная сумма пакета, рассчитывается для байтов 2–15. Вычисление производится с использованием полинома 0xB5.

Passw – пароль доступа к информации. Определяются два пароля с уровнями доступа соответственно ADM и USR. Пользователь ADM имеет возможность записи и считывания данных и конфигурации. Пароль ADM равен 777777 (байты в слове наоборот).

Serv – сервисное поле (рис. 4.10), определяющее следующие функции: Direct – тип обмена данными (1 – запрос, 0 – ответ); ClassAccess – класс доступа к счетчику (0x05 – выполнение команды); MessageLength – количество байтов, которые помещены в данные, (2 байта).

Значение	Direct	ClassAccess			MassageLength			
№ бита	7	6	5	4	3	2	1	0
Значение	0	1	0	1	0	0	1	0

Рис. 4.10. Структура сервисного поля при запросе

AddrH, AddrL – старший и младший байты кода команды счетчику.

Data – данные.

Пример команды запроса для считывания суммарной электрической энергии приведен на рис. 4.11.

Структура сообщения при ответе протокола CE показана на рис. 4.12.

Команда [hex код]	Описание команды (адресные)	Значение по умолчанию	Запрос [количество передаваемых байт]	Ответ [количество принимаемых байт]
ReadMonthEnergy [0x0130]	Чтение значения энергии, сохраненное НА конец месяца/ЗА месяц.		[2 байта] UINT8[1] – индекс глубины опроса (0 – текущие значения, 1 – за прошедший месяц, 2 – два месяца назад, … 12 – 12 месяцев назад) , BIT[7]: 0 – НА, 1 – ЗА; UINT8[2] – номер тарифа (0 – сумма по тарифам, 1…5 – тариф 1…5).	[7 байт] BCD[1] – день BCD[2] – месяц BCD[3] – год DATA4[1] – значение энергии.

Рис. 4.11. Структура команды запроса на чтение значения энергии

Отметим, что в поле Data значение энергии передается младшими байтами вперед.

Поле	END	OPT	AddrS		AddrD		Serv	AddrH	AddrL	Data (Дата)			Data (кВт)				CRC8	END
№ байта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Значение	C0	48	FD	00	69	91	57	01	30	03	03	23	35	00	00	00	57	C0

Рис. 4.12. Структура сообщения при ответе протокола CE

По второй шине RS-485 контроллер является подчиненным устройством (Slave). Мастером (Master) может быть одно из трех устройств: конвертер USB/RS-485, преобразователь интерфейсов MOXA или сетевой шлюз ОВЕН ПВ-210. Поскольку в сети может быть только один Master, в данной шине предусмотрен переключатель (SA1).

Далее, в разд. 4.2.3, рассмотрен пример построения сети с ПК по протоколу Modbus RTU посредством конвертора USB/RS-485. В контроллере прописаны параметры шины RS-485 № 2, которые идентичны параметрам шины RS-485 № 1 (см. данные на с. 97). Выбраны эти настройки, т. к. они являются самыми распространенными. В данном случае ничего не мешает поменять параметры в контроллере.

4.2.3. Подготовка к работе

Учебный стенд необходимо в следующем порядке подготовить к работе.

– Переключатель SA1 установить в среднее положение (положение 0).

– Включить шнур питания (W1) в розетку.

⚠ В розетке обязательно должна присутствовать клемма заземления.

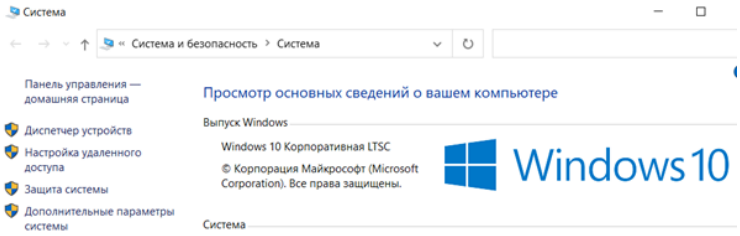
– Подключить к ПК провод USB с маркировкой «RTU».

– Включить все автоматические выключатели (QF0, QF1, QF2, QF3).

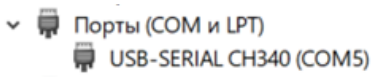
4.3. Настройки в системе Windows

⚠ Для автоматической установки драйвера нужно обязательно подключить ПК к интернету!

Заходим в **Мой компьютер**, нажимаем **ПКМ** и **Свойства**.



Далее заходим в **Диспетчер устройств**, там должен быть пункт меню **Порты (COM и LTP)**. Если такого нет, стоит подождать, папка должна обновиться автоматически; это значит, что драйвера еще не установились. Раскрываем этот пункт меню и находим устройство **USB-SERIAL CH340**. В скобках указано, к какому COM-порту подключено устройство; данная информация пригодится для дальнейшей настройки. В нашем случае это COM5, в вашем это значение может отличаться.



Два раза щелкаем по устройству и переходим во вкладку **Параметры порта** (рис. 4.13). Нажимаем **ОК** и закрываем открытые папки.

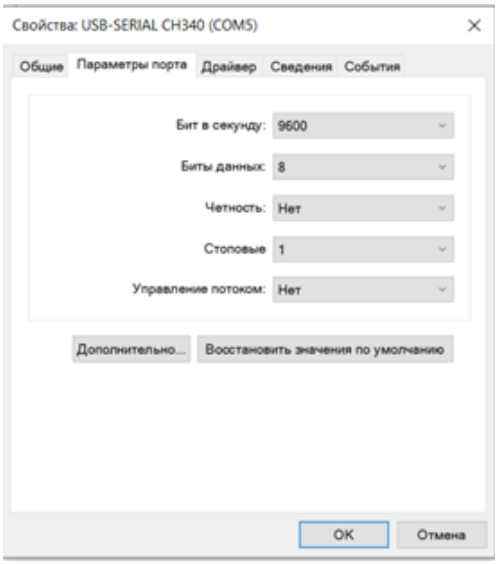


Рис. 4.13. Окно «Свойства USB-SERIAL CH340(COM5)»

4.4. Обзор переменных контроллера для удаленного управления

В программе на контролере используются переменные (табл. 19). Они необходимы для удаленного управления оборудованием с ПК.

Таблица 19

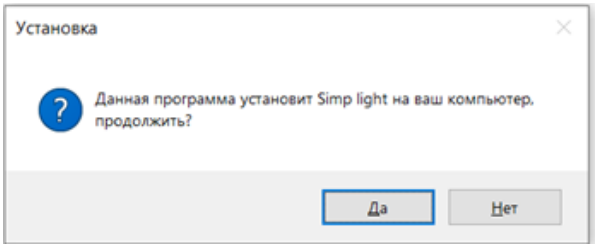
Переменные контроллера

Название	**Адрес**	**Регион памяти**	**Тип переменной**	**Тип доступа**
Вход №1	0x00	Discrete Input	Bool	ReadOnly
Вход №2	0x01	Discrete Input	Bool	ReadOnly
Вход №3	0x02	Discrete Input	Bool	ReadOnly
Вход №4	0x03	Discrete Input	Bool	ReadOnly
Вход №5	0x04	Discrete Input	Bool	ReadOnly
Вход №6	0x05	Discrete Input	Bool	ReadOnly
Вход №7	0x06	Discrete Input	Bool	ReadOnly
Вход №8	0x07	Discrete Input	Bool	ReadOnly
Выход №1	0x00	Coils	Bool	ReadWrite
Выход №2	0x01	Coils	Bool	ReadWrite
Выход №3	0x02	Coils	Bool	ReadWrite
Выход №4	0x03	Coils	Bool	ReadWrite
Счётчик воды	0x090A	Holding Register	DWord	ReadOnly
Счётчик эл. энергии	0x0B0C	Holding Register	DWord	ReadOnly
Управление ПЧ Вперед – 100 Назад – 200 Стоп – 600	0x0D	Holding Register	Word	ReadWrite
Скорость ПЧ (0–5000)	0x0E	Holding Register	Word	ReadWrite

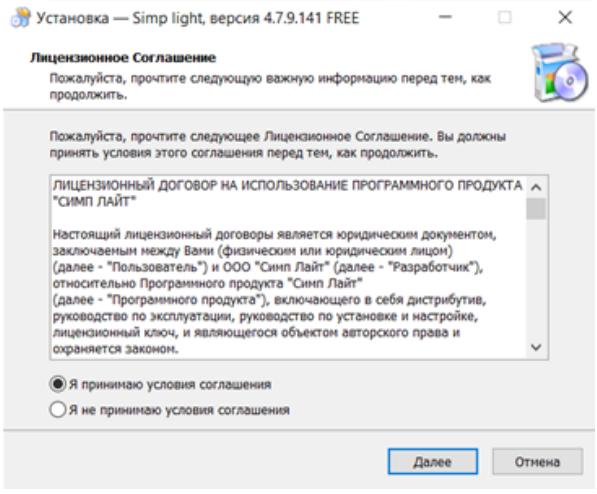
Для демонстрации работы сети может использоваться любое ПО на ПК, предназначенное для контроля и настройки устройств, использующих протокол Modbus RTU. Ниже рассмотрен пример работы с отечественной системой SCADA *Simp light* [13].

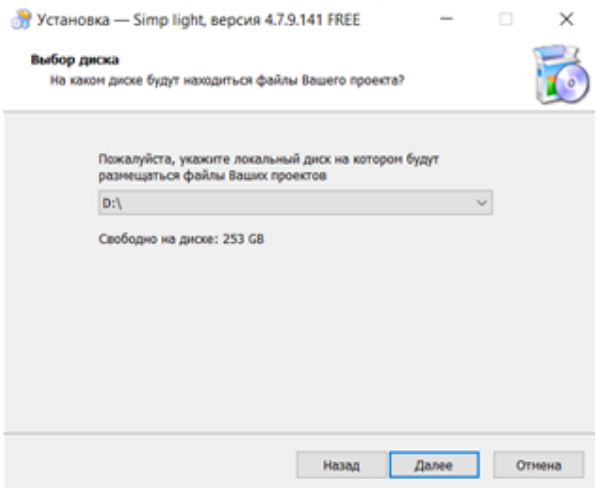
4.4.1. Установка системы SCADA

Прежде всего необходимо установить SCADA-систему на ПК. Эта программа есть в открытом доступе на официальном сайте [13]. Там же, в разделе «Техподдержка», есть онлайн-документация, где подробно показано, как работать с системой Simp light [14]. Открываем ярлык дистрибутива **Simp light**. Нажимаем **Да**.

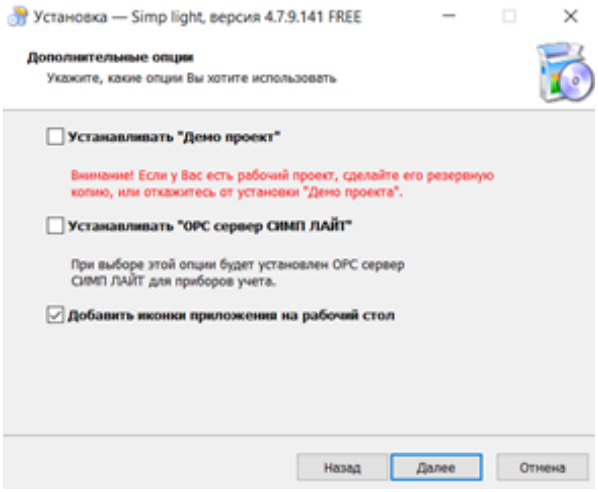


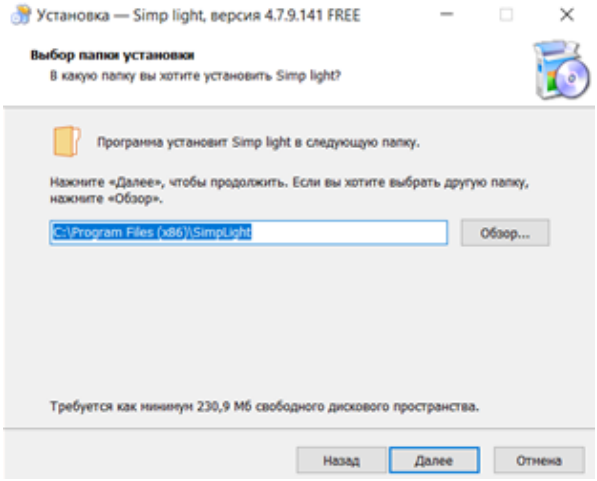
Принимаем условия и нажимаем **Далее**. Выбираем нужный диск, куда будут сохраняться проекты.



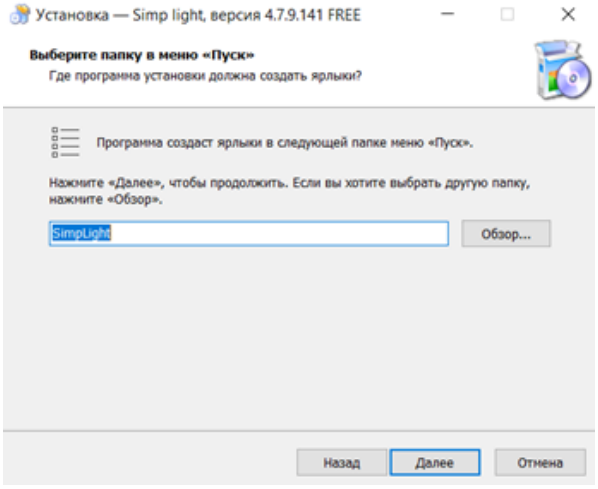


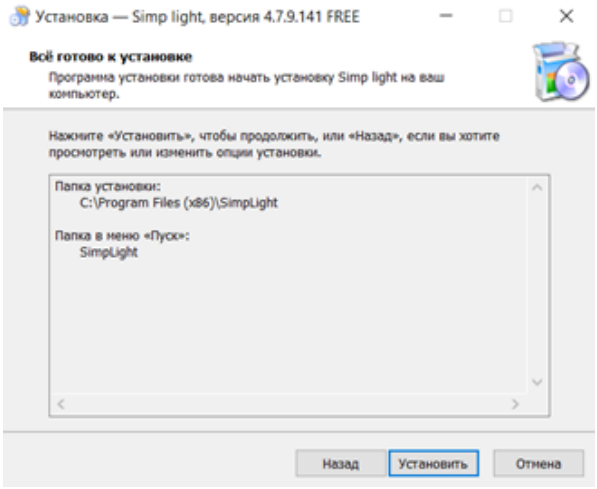
Далее выбираем только **Добавить иконки приложения на рабочий стол** и нажимаем **Далее**. Выбираем место, куда будет установлена программа, и нажимаем **Далее**.





Нажимаем **Далее**. Нажимаем **Установить**.

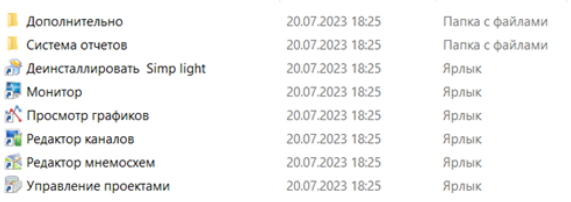




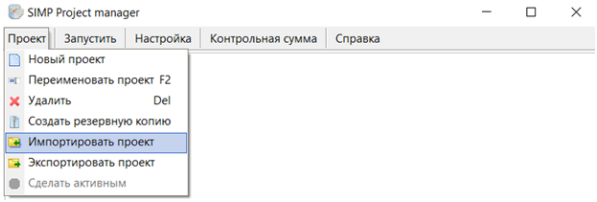
После установки появится иконка на рабочем столе. Процесс установки закончен.

4.4.2. Настройки в системе SCADA

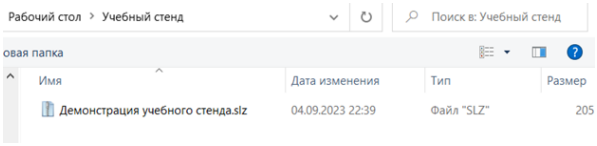
В данном случае мы будем использовать готовый проект для демонстрации. Для этого нажимаем на иконку **SCADA** – появляется папка с ярлыками.



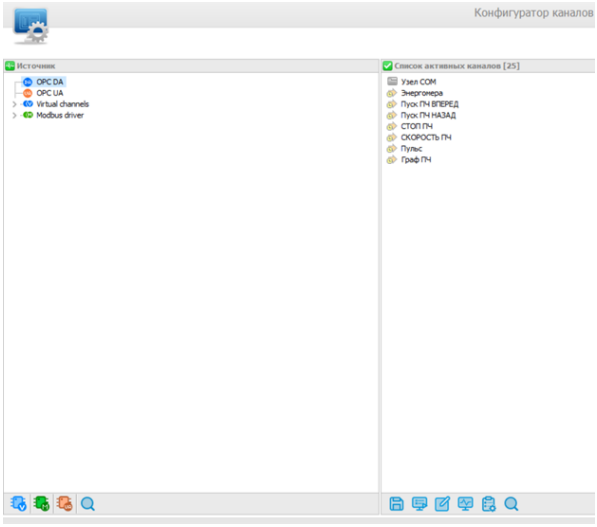
Открываем ярлык **Управление проектами** и нажимаем **Импортировать проект**.

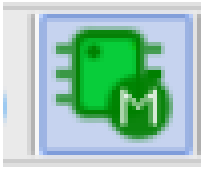


Выбираем путь, где сохранен готовый проект.

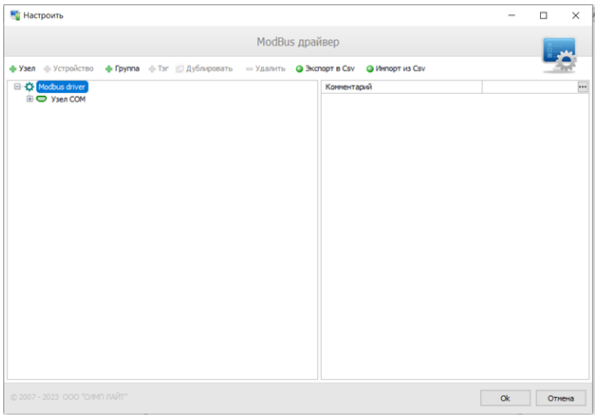


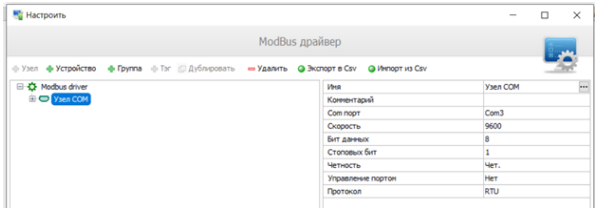
Импортируем проект и открываем его, дважды нажав на **ЛКМ**. Появляется основное окно SCADA-системы. В левом нижнем углу нажимаем на зеленую иконку «Настройка ModBus драйвера».





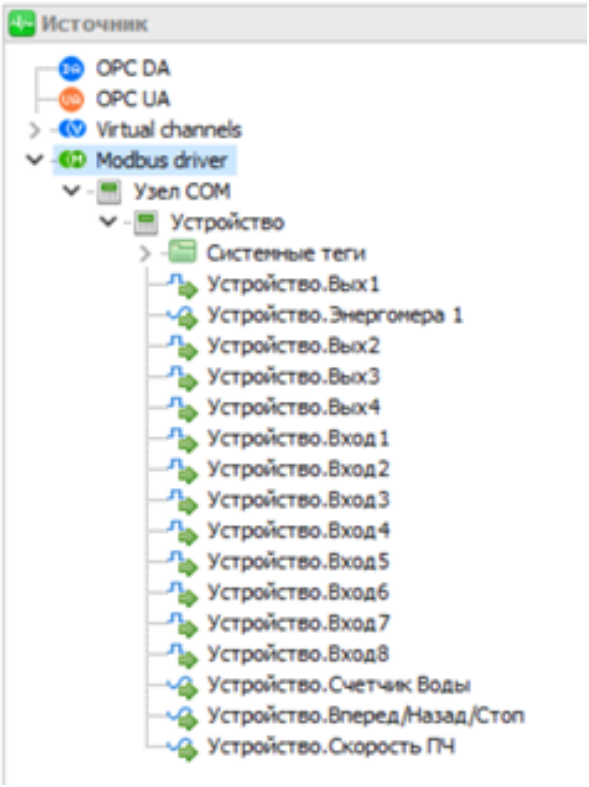
Появляется окно настройки. Далее нажимаем на **Узел COM**.





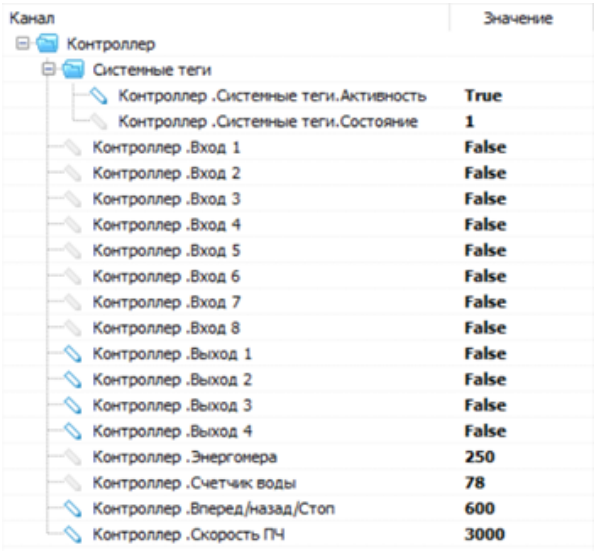
В пункте **COM порт** выбираем то значение, которое было указано в «Диспетчере устройств». Остальные параметры оставляем без изменений.

Нажимаем **ОК** и видим в пункте меню **Источник** наши переменные.

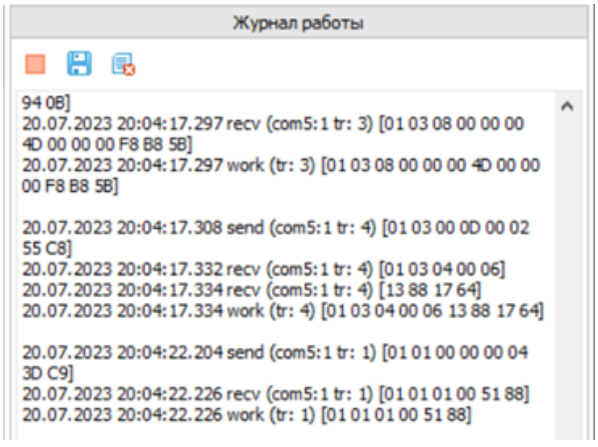


Далее необходимо проверить, есть ли связь между SCADA и контроллером. Нажимаем на кнопку «Тест каналов».

Появляется окно «Тест каналов». Для того чтобы определить наличие связи между ПК и контроллером, смотрим на **Системный тег.Состояние**. Если он равен 1, то ПК и контроллер обмениваются информацией – связь есть.



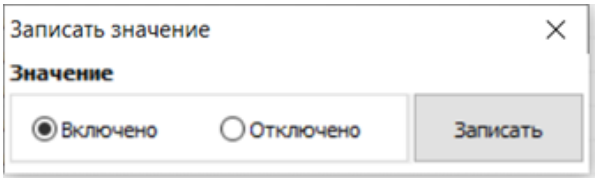
Остальные переменные показывают те значения, которые в данный момент фигурируют в контроллере. Справа есть окно журнала работы. Если нажать на кнопку **Синяя стрелка**, то можно увидеть в режиме реального времени, какие слова запроса отправляет SCADA на контроллер и что получает в ответ.



4.4.3. Проверка работы в системе SCADA

Выход контроллера

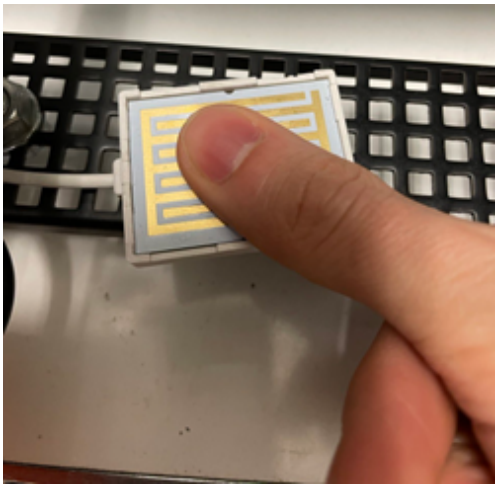
В окне отладки можно записывать новые значения в переменные. Для примера включим лампочку, которая подключена к выходу № 2 на контроллере. Для этого два раза щелкаем ЛКМ по значению **False** напротив тега «Выход 2» (адрес – 0x01). Нажимаем на значение **Включено** и **Записать**.



В окне отладки должно появиться значение «True». На стенде сработает реле KL2 и загорится лампочка HL2.

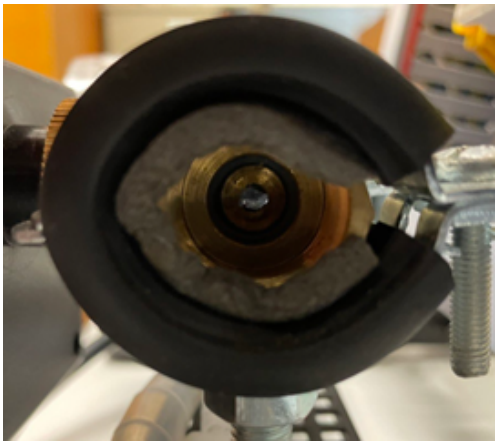
Датчик протечки воды и задвижка

Проверим работу датчика протечки воды (LS1). Для этого достаточно намочить палец и приложить его к датчику.

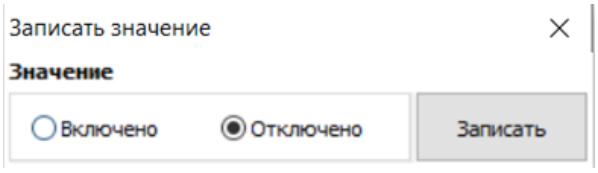


После того как датчик сработал, переменные «Выход 1» и «Вход 5» поменяют свои значения на «True». На стенде сработает реле KL1.

Задвижка G1 начнет закрываться. В течение 15 с она полностью перекроет поток. ⚠ До того как открыть задвижку, убедитесь, что она полностью закрыта. Если она всё еще работает, дождитесь полного закрытия.

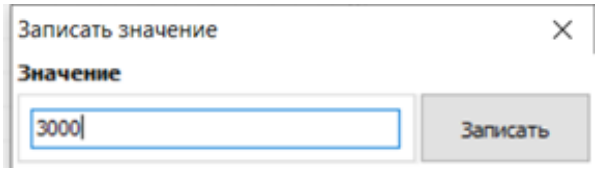


Для того чтобы открыть задвижку, нужно отключить реле KL1. Для этого нажимаем на переменную **Выход 1** и записываем значение **Отключено**.



Преобразователь частоты

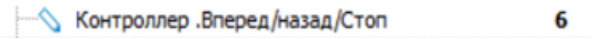
Введем задание скорости на преобразователь частоты. Запишем значение в диапазоне 1–5000. В нашем случае 3000 (30 Гц). Вы можете задать любое другое значение.



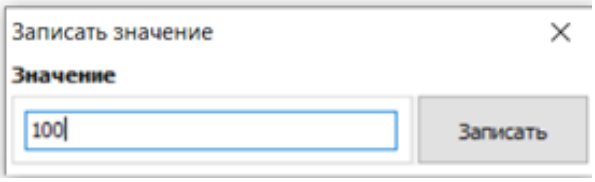
Задание должно поменяться на то значение, которое вы записали.

На преобразователе частоты должно быть такое же значение, как и в системе SCADA.

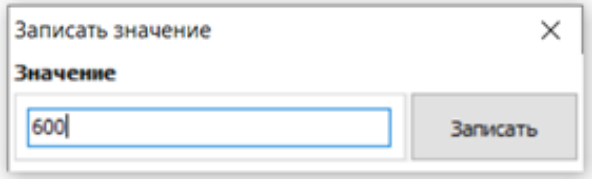
Переменная с адресом 0x0D отвечает за пуск, реверс и остановку двигателя (как указано в табл. 19).



Запишем в эту переменную значение 100 и нажмем на кнопку **Записать** – двигатель начнет вращаться с той скоростью, которую мы задали ранее.



Запишем в эту переменную значение 600 и нажмем кнопку **Записать** – двигатель остановится.



Счетчик электрической энергии

Для демонстрации передачи значения электрической энергии (переменная 0x0B) от счетчика достаточно подключить нагрузку (чайник, наполненный водой) в розетку (XS1) на стенде и включить чайник. В течение 30 с значение должно одновременно поменяться на счетчике и на ПК.

Расходомер воды

Для демонстрации передачи значения от счетчика воды (переменная 0x09) достаточно путем воздушного нагнетания через шланг воздействовать на крыльчатку счетчика. В диапазоне 0,001–0,003 м^3 на счетчике срабатывает геркон, тем самым оправляя импульс на вход контроллера (I1). Значение переменной должно измениться на 1.

Запуск рабочего монитора

Нужно нажать на иконку **Запустить монитор**. Далее появится рабочий монитор, на котором можно видеть показания счетчиков и статуса задвижки, а также управлять преобразователем частоты.

Конрольные темы для повторения

1. Функциональная возможности учебно-демонстрационного комплекса приёма-передачи данных.
2. Привести варианты датчиков для подключения к дискретным входам ПЛК.
3. Привести варианты устройств для подключения к дискретным выходам ПЛК.
4. Привести варианты устройтств для подключения к шине ModBus.
5. Целесообразность приенения ModBus и CAN в системах малой автоматизации и автоматизации зданий и сооружений.

Список библиографических ссылок

1. Микродат. О компании : [сайт]. – URL: https://www.microdat.ru/about.php (дата обращения: 20.10.2023). – Загл. с титул. экрана.
2. IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages : [сайт]. – URL: https://webstore.iec.ch/publication/4552 (дата обращения: 20.10.2023). – Загл. с титул. экрана.
3. Хоровиц П., Хилл У. Искусство схемотехники : пер. с англ. – Изд. 2-е. – М. : БИНОМ, 2014. – 704 с.
4. STM32F072x8, STM32F072xB : [pdf-страница]. – URL: https://www.st.com/content/ccc/resource/technical/document/datasheet/cd/46/43/83/22/d3/40/c8/DM00090510.pdf/files/DM00090510.pdf/jcr:content/translations/en.DM00090510.pdf (дата обращения: 20.10.2023). – Загл. с титул. экрана.
5. STM32CubeIDE Integrated Development Environment for STM32 : [сайт]. – URL: https://www.st.com/en/development-tools/stm32cubeide.html (дата обращения: 20.10.2023). – Загл. с титул. экрана.
6. Description of STM32F0 HAL and low-layer drivers : [pdf-страница]. – URL: https://www.st.com/resource/en/user_manual/dm00122015-description-of-stm32f0-hal-and-lowlayer-drivers-stmicroelectronics.pdf (дата обращения: 20.10.2023). – Загл. с титул. экрана.
7. Нестеров К.Е., Зюзёв А.М. Программирование промышленных контроллеров : учеб.-метод. пособие. – Екатеринбург : Изд-во Урал. ун-та, 2019. – 96 с.
8. Modbus : [сайт]. – URL: https://modbus.org/ (дата обращения: 20.10.2023). – Загл. с титул. экрана.
9. MODBUS over Serial Line Specification and Implementation Guide V1.02 : [pdf-страница]. – URL: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (дата обращения: 20.10.2023). – Загл. с титул. экрана.
10. MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3 : [pdf-страница]. – URL: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf(дата обращения: 20.10.2023). – Загл. с титул. экрана.
11. Road vehicles Controller area network (CAN) Part 1: Data link layer and physical signalling : [сайт]. – URL: https://www.iso.org/standard/63648.html (дата обращения: 20.10.2023). – Загл. с титул. экрана.

12. CiA : [сайт]. – URL: https://www.can-cia.org/ (дата обращения: 20.10.2023). – Загл. с титул. экрана.
13. SCADA Simplight : [сайт]. – URL: https://simplight.ru/ (дата обращения: 20.10.2023). – Загл. с титул. экрана.
14. Быстрый старт / Создание нового проекта : [сайт]. – URL: https://simplight.ru/manual_next/bystryy-start/sozdanie-novogo-proekta (дата обращения: 20.10.2023). – Загл. с титул. экрана.

Учебное издание

Зюзев Анатолий Михайлович
Мудров Михаил Валентинович
Костылев Алексей Васильевич

ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ КОНТРОЛЛЕРЫ В СИСТЕМАХ МАЛОЙ АВТОМАТИЗАЦИИ И АВТОМАТИЗАЦИИ ЗДАНИЙ И СООРУЖЕНИЙ

Редактор *И. В. Меркурьева*
Компьютерная верстка в LaTeX *М. В. Мудрова*

Подписано в печать 29.11.2023. Формат 70×100/16.
Бумага писчая. Печать цифровая. Усл. печ. л. 9,7.
Уч.-изд. л. 8,3. Тираж 50 экз. Заказ 197.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ ЦСД
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел. 8(343) 375-48-25, 375-46-85, 374-19-41
E-mail rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: +7 (343) 358-93-06
http://print.urfu.ru

Для заметок

9 785799 637880

ЗЮЗЕВ АНАТОЛИЙ МИХАЙЛОВИЧ
Доктор технических наук, профессор каф. «Электропривод и автоматизация промышленных установок» УралЭНИН УрФУ.
Область исследований — электротехнические комплексы и системы, автоматизированный электропривод отраслевого применения (нефтегазовый комплекс, транспортные системы и пр.). Сфера научных интересов: Имитационное моделирование электроприводов и технологических машин и механизмов, в т. ч. моделирование в реальном времени; анализ и оптимизация режимов работы электропривода; разработка экспертных систем для оценки состояния машинных агрегатов на основе переменных электропривода.

МУДРОВ МИХАИЛ ВАЛЕНТИНОВИЧ
Кандидат технических наук, доцент кафедры «Электропривод и автоматизация промышленных установок», УралЭНИН, УрФУ. Сфера научных интересов: программно-аппаратные комплексы реального времени для испытаний электроприводов, современные электроприводы с цифровым управлением.

КОСТЫЛЕВ АЛЕКСЕЙ ВАСИЛЬЕВИЧ
Кандидат технических наук, доцент, заведующий кафедрой «Электропривод и автоматизация промышленных установок», УралЭНИН, УрФУ.
Область научных интересов: асинхронные электроприводы с полупроводниковыми преобразователями частоты, системы автоматизации зданий и сооружений.