**Белорусский национальный технический университет**
Факультет информационных технологий и робототехники
Кафедра «Робототехнические системы»

СОГЛАСОВАНО
Заведующий кафедрой
_____Г.Н. Здор
_ _____ 2016 г.

СОГЛАСОВАНО
Декан факультета
_____Е.Е. Трофименко
__ _____ 2016 г.

**УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ**

# Программное управление технологическим оборудованием

для специальности:
I – 53 01 01 «Автоматизация технологических процессов и производств»

Составитель: Сиротин Ф.Л.

Рассмотрено и утверждено
на заседании совета факультета информационных технологий и робототехники мая 2016 г.,
протокол N

**СОДЕРЖАНИЕ**

# I. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

# 1. ПРОГРАММИРОВАНИЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ КОНТРОЛЛЕРОВ

## 1.1 Назначение и структура программируемого контроллера

Программируемый логический контроллер (ПЛК) − специализированное микропроцессорное устройство со встроенным аппаратным и программным обеспечением, которое используется для выполнения функций управления технологическим оборудованием. Прародителями ПЛК были релейные схемы автоматики. Это "родство" до сих пор проявляется в виде жесткой цикличности выполнения программы и своеобразного языка программирования. ПЛК – устройство, доступное для программирования неспециалисту в области информатики и предназначенное для управления последовательными логическими процессами в условиях промышленной среды в реальном масштабе времени. ПЛК циклически опрашивает входы, к которым подключены выключатели, датчики и.т.д., и в зависимости от их состояния («включено» − 1, «выключено» − 0), включает-выключает выходы, а следовательно и подключенные к выходам исполнительные механизмы. Функциональная схема системы управления (СУ) на базе контроллера показана на рисунке 1.1. Используя программное обеспечение, пользователь имеет возможность программировать контроллер или вносить изменения в уже существующую программу.

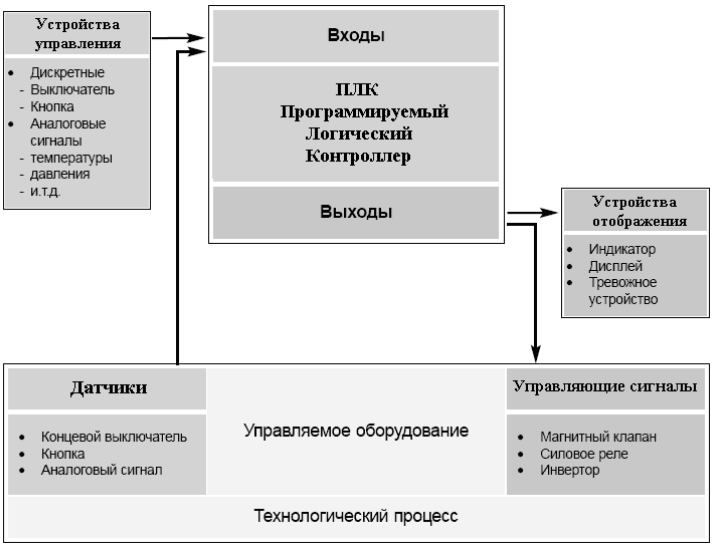



Рис. 1.1. Функциональная схема СУ на базе ПЛК

Программируемый логический контроллер, главным образом состоит из центрального процессора (ЦП), области памяти и функций обработки сигналов ввода/вывода (т.е., входов и выходов). Условно можно назвать такой

контроллер основным, или базовым блоком (модулем). Можно считать, что ПЛК − это сотни или тысячи отдельных реле, счетчиков, таймеров и память. Все эти счетчики, таймеры моделируются ЦП и осуществляют логику работы согласно заложенной программы. Структурная схема контроллера показана на рисунке 1.2.

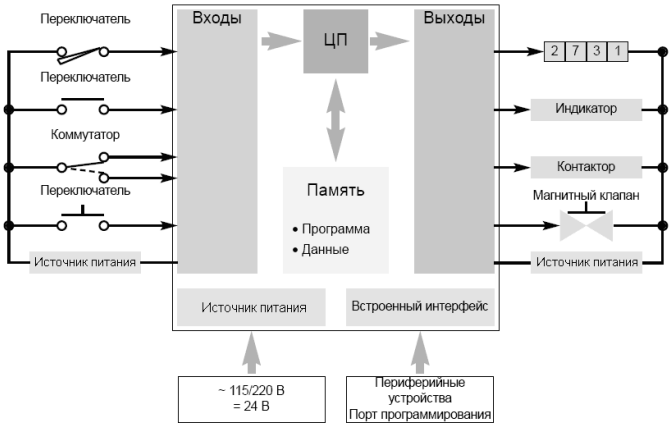



Рис. 1.2. Структурная схема контроллера

•*ВХОДЫ* обеспечивают связь с внешними устройствами. Физически существуют и получают сигналы от выключателей, датчиков, и т.д. Различают аналоговые и цифровые входы, предназначенные для работы с аналоговыми и цифровыми сигналами соответственно.

•*ЦП* – «мозг» ПЛК, осуществляющий логику работы системы. Это процессор, обрабатывающий команды программы и управляющий всеми внутренними элементами контроллера: входами, выходами, счетчиками, таймерами, внутренними реле, регистрами и т.д. На рисунке 1.2 счетчики, таймеры и внутренние реле не показаны отдельно, они входят в состав микросхемы ЦП. Т.е. каждый контроллер обладает фиксированным набором таких элементов, которые приводятся в спецификации.

•*ВНУТРЕННИЕ РЕЛЕ (МЕРКЕРЫ)* предназначены для обеспечения работы программы, т.к. являются своего рода единицами хранения информации. Наряду с обычными меркерами существуют также и служебные меркеры, несущие специальную смысловую и функциональную нагрузку (например, установка разрешающего флага для запуска высокоскоростных счетчиков). Назначение каждого конкретного служебного меркера приводится в документации к контроллеру.

•*СЧЕТЧИКИ* предназначены для различного рода счета. Отдельно выделяют высокоскоростные счетчики. Как правило, имеются ограничения на скорость счета и значение, до которого ведется счет, для чего необходимо обращаться к документации конкретного контроллера.

•*ТАЙМЕРЫ* предназначены, как правило, для установки времени задержки включения/выключения и т.п. Различаются в основном точностью отсчета времени и, как следствие, назначением.

•*ПАМЯТЬ* – контроллер обладает некоторым объемом памяти, которая в различных контроллерах может иметь различную организацию. Как правило, память делится на рабочую область (ОЗУ), куда загружается программа непосредственно во время работы контроллера, и область данных (EEPROM, MMC и т.п), где хранится программа и различные данные. Часто объем рабочей области измеряется в килобайтах, а объем области данных – в количестве шагов программы.

•*ВСТРОЕННЫЙ ИНТЕРФЕЙС* обеспечивает подключение ПЛК к компьютеру или программатору для обмена данными, в том числе и для перепрограммирования контроллера. В основном это RS-232C (COM-port), RS-422, RS-485 и т.п.

•*ВЫХОДЫ* обеспечивают связь с внешними устройствами, т.е. обеспечивают включение/ выключение исполнительных механизмов. Существуют два варианта исполнения: релейные, полупроводниковые (транзисторные и симисторные). Различают аналоговые и цифровые выходы, предназначенные для работы с цифровыми и аналоговыми сигналами соответственно.

•*ИСТОЧНИК ПИТАНИЯ* предназначен для обеспечения работы контроллера. Могут использоваться внешние источники питания, как постоянного тока +12/24 В, так и переменного – ~110/220 В. Многие контроллеры обладают встроенными сервисными источниками питания (обычно +12/24 В), которые используются для подачи питания на датчики или другие устройства, подключенные к контроллеру для упрощения входных и выходных цепей..

Последнее время имеется тенденция к расширению функциональных возможностей контроллеров за счет реализации встроенных ПИД-регуляторов, часов реального времени, объединения контроллеров в сеть и использования возможностей подключения блоков расширения. В любом случае структура контроллера остается неизменной, и выбор модели определяется только требованиями технологического процесса, а широкий ряд моделей позволяет подобрать контроллер с оптимальным соотношением цена/производительность. Вопросы, связанные с выбором контроллера рассмотрены в разделе 6 «Вопрос выбора ПЛК».

Для понимания работы контроллера на рисунке 1.3 приведен алгоритм его работы.

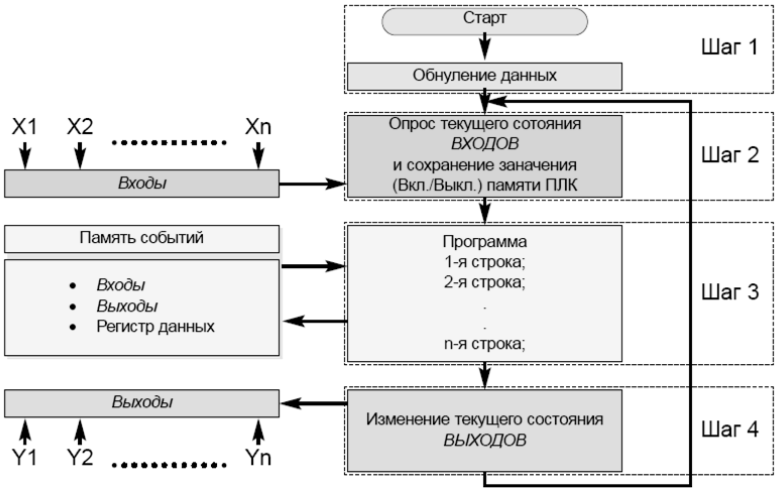



Рис. 1.3. Схема алгоритма работы контроллера

В процессе работы ПЛК непрерывно опрашивает текущее состояние входов X1, X2…Xn и в соответствии с требованиями производственного процесса изменяет состояние выходов Y1, Y2…Yn (вкл./выкл). Можно разделить этот цикл на четыре основных шага.

Шаг первый – инициализация системы. Необходимо помнить, что в случае сбоев по питанию или при выключении контроллера система обязана вернуться в исходное состояние. Не следует недооценивать важности этой части программного кода, так как в противном случае это может привести к сбоям и поломкам оборудования.

Шаг второй – проверка текущего состояния входов. ПЛК проверяет текущее состояние входов и в зависимости от их состояния («вкл». или «выкл.») выполняет последовательные действия, указанные в программе. Состояние любого из входов сохраняется в памяти (в области данных) и может в дальнейшем использоваться при обработке третьего шага программы.

Шаг третий – выполнение программы. Будем считать, что в ходе технологического процесса переключился вход (X1) с «выключено» на «включено», и в соответствии с технологическим процессом нам необходимо изменить текущее состояние выхода (Y1) с «выключено» на «включено». Так как ЦП опросил текущее состояния всех входов и хранит их текущее состояние в памяти, то выбор последующего действия обусловлен только ходом технологического процесса.

Шаг четвертый – изменение текущего состояния выхода. ПЛК изменяет текущее состояние выходов в зависимости от того, какие входы являются выключенными, а какие включенными, исходя из алгоритма записанной в память программы, которая была отработана на третьем шаге. То есть контроллер, физически переключил выход (Y1) и включились исполнительные механизмы: лампочка, двигатель и т.д. После этого следует возврат на второй шаг.

## 1. 2 Классификация контроллеров

Как правило, ПЛК объединяет в себе базовый блок и широкий спектр модулей расширения, позволяющих сконфигурировать оптимальную систему непосредственно под каждую конкретную задачу. В зависимости от данной задачи можно выбрать как небольшие и недорогие не наращиваемые контроллеры (которые не поддерживают подключение модулей расширения), так и масштабируемые решения с возможностью подключения дополнительных модулей с обширным набором возможностей. Невозможно сказать, какой из контроллеров лучше, а какой хуже; широкий ряд контроллеров позволяет решать задачи с оптимальным соотношением цена-производительность, и выбор конкретной модели определяется только требованиями, которые предъявляются решаемой задачей (память, быстродействие, возможность расширения, необходимость создания сети).

Мощное вычислительное ядро современных ПЛК делает их очень похожими на компьютеры. Однако ПЛК это не «железо», а технология. Она включает специфическую аппаратную архитектуру, принцип циклической работы и специализированные языки программирования. Программирование ПЛК осуществляется людьми, хорошо знающими прикладную область, но не обязанными быть специалистами в математике и программировании.

В настоящее время в промышленности используется несколько типов логических контроллеров.

1) **Встроенный**, являющийся неотъемлемой частью агрегата, машины, прибора. Такой контроллер может управлять станком с ЧПУ, современным интеллектуальным аналитическим прибором, автомашинистом и другим оборудованием. Выпускается на раме без специального кожуха, поскольку монтируется в общий корпус оборудования. Такой контроллер может быть как однокристальным, так и представлять собой набор плат и интегральных схем без собственного корпуса.

2) **Автономный модуль**, реализующий функции контроля и управления небольшим изолированным технологическим узлом, как, например, районные котельные, электрические подстанции, резервуарные парки. Автономные контроллеры помещаются в защитные корпуса, рассчитанные на разные условия окружающей среды. Почти всегда эти контроллеры имеют порты для соединения в режиме «точка-точка» (peer-to-peer) с другой аппаратурой и интерфейсы, связывающие отдельные устройства через сеть с другими средствами автоматизации. В контроллер встраивается или подключается к нему специальная панель интерфейса для оператора, состоящая из алфавитно-цифрового дисплея и набора функциональных клавиш.

В этом классе следует выделить специальный тип локальных контроллеров, предназначенных для систем противоаварийной защиты. Такие устройства отличаются особенно высокой надежностью и быстродействием. В них предусматриваются различные варианты полной текущей диагностики

неисправностей, вплоть до диагностики неисправностей каждой отдельной платы; защитные коды, предохраняющие информацию от искажений во время передачи и хранения; резервирование как отдельных компонентов, так и всего устройства в целом.

Контроллеры, предназначенные для цепей противоаварийной защиты, должны иметь специальный сертификат, подтверждающий их высокую надежность и устойчивость к внешним воздействиям.

3) **Сетевой комплекс контроллеров (PLC,** Network).

Сетевые решения на базе ПЛК наиболее широко применяются для управления производственными процессами во всех отраслях промышленности. Минимальный состав данного класса ПТК подразумевает наличие следующих компонентов:

- набор контроллеров;
- несколько дисплейных рабочих станций операторов;
- системную (промышленную) сеть, соединяющую контроллеры между собой и контроллеры с рабочими станциями.

Контроллеры каждого сетевого комплекса, как правило, имеют ряд модификаций, отличающихся друг от друга быстродействием, объемом памяти, возможностями по резервированию, способностью работать в разных условиях окружающей среды, числом каналов входа/выхода. Так что можно подобрать контроллер для каждого узла автоматизируемого агрегата с учетом особенностей и выполняемых функций последнего и использовать один и тот же комплекс для управления разными производственными объектами.

Следует выделить телемеханический тип сетевого комплекса контроллеров, предназначенный для автоматизации объектов, распределенных по большой области пространства.

Системная сеть с характерной структурой и особые физические каналы связи (радиоканалы, выделенные телефонные линии, оптоволоконные кабели) позволяют интегрировать узлы объекта, отстоящие друг от друга на многие десятки километров, в единую систему автоматизации.

Рассматриваемый класс сетевых комплексов контроллеров имеет верхние ограничения как по сложности выполняемых функций, так и по объему автоматизируемого объекта. Обычно телемеханические комплексы решают типовые задачи измерения, контроля, учета, регулирования и блокировки, учитывая до нескольких десятков тысяч измеряемых и контролируемых величин.

Чаще всего сетевые комплексы применяются на уровне цехов машиностроительных заводов, агрегатов нефтеперерабатывающих, нефтехимических и химических производств (правда, не самых мощных), а также цехов предприятий пищевой промышленности. Телемеханические сетевые комплексы контроллеров используются для управления газо- и нефтепроводами, электрическими сетями, транспортными системами.

4) **Распределенные маломасштабные системы управления (DCS,** Smaller Scale).

Маломасштабные распределенные контроллерные средства в среднем превосходят большинство сетевых комплексов контроллеров по мощности и гибкости структуры, а следовательно, по объему и сложности выполняемых функций. В целом, этот класс еще имеет ряд ограничений по объему автоматизируемого производства и набору реализуемых функций. Однако данная категория средств отличается от предшествующего класса тем, что имеет развитую многоуровневую сетевую структуру. Так, нижний уровень может выполнять связь контроллеров и рабочей станции компактно расположенного технологического узла, а верхний уровень поддерживать взаимодействие нескольких узлов друг с другом и с рабочей станцией диспетчера всего автоматизируемого участка производства. На верхнем уровне (уровне рабочих станций операторов) эти комплексы, по большей части, имеют достаточно развитую информационную сеть. В некоторых случаях расширение сетевой структуры идет в направлении применения стандартных цифровых полевых сетей, соединяющих отдельные контроллеры с удаленными от них блоками ввода/вывода и интеллектуальными приборами. Подобная простая и дешевая сеть соединяет по одной витой паре проводов контроллер с множеством интеллектуальных полевых (заводских) приборов, что резко сокращает длину кабельных сетей на предприятии и уменьшает влияние возможных помех, поскольку исключается передача низковольтной аналоговой информации на значительные расстояния.

Мощность контроллеров, применяемых в этом классе средств, позволяет в дополнение к типовым функциям контроля и управления реализовывать более сложные и объемные алгоритмы управления (например, самонастройку алгоритмов регулирования, адаптивное управление).

Маломасштабные распределенные системы управления используются для автоматизации отдельных средних и крупных агрегатов предприятий непрерывных отраслей промышленности, а также цехов и участков дискретных производств и цехов заводов черной и цветной металлургии.

5) **Полномасштабные распределенные системы управления (DCS, Full** Scale).

Это наиболее мощный по возможностям и охвату производства класс контроллерных средств, практически не имеющий границ ни по выполняемым на производстве функциям, ни по объему автоматизируемого производственного объекта. Нередки примеры использования одной такой системы для автоматизации производственной деятельности целого крупномасштабного предприятия.

Описываемая группа контроллерных средств отличается:

− развитой многоуровневой структурой, предусматривающей выделение трех уровней: информационного, системного и полевого, причем

для организации отдельных уровней могут использоваться разные варианты построения сетей;

- клиент-серверным режимом работы;
- выходом на корпоративную сеть предприятия, систему управления бизнес-процессами, глобальную сеть Интернет, а также на уровень интеллектуальных приборов;
- широким модельным рядом применяемых контроллеров, различающихся по числу входов/выходов, быстродействию, объему памяти разного типа, возможностям по резервированию, наличию встроенных и удаленных интеллектуальных блоков ввода/вывода на все виды аналоговых и дискретных сигналов;
- широким диапазоном рабочих станций;
- мощным современным программным обеспечением, в состав которого входят:

а) интерфейсы операторов с системой управления, предусматривающие различные варианты построения на разных уровнях управления;

б) набор технологических языков с объемными библиотеками типовых программных модулей для решения задач контроля, логического управления и регулирования;

в) универсальные прикладные пакеты программ, реализующие типовые функции управления отдельными агрегатами, диспетчерское управление участками производства, технический учет и планирование производства в целом,

г) системы автоматизированного проектирования и конструкторского документооборота для разработки системы автоматизации.

Полномасштабные распределенные системы управления устанавливаются на электростанциях, крупных агрегатах типа «котел-турбина», нефтеперерабатывающих заводах для управления крекинг-процессами, охватывают все производство на химических и нефтехимических заводах и т. д.

К перечисленным выше видам контроллеров можно добавить также то, что существуют программы, имитирующие работу ПЛК на компьютере, так называемые SoftPLC (программные ПЛК). В этом случае, удается совместить на одной машине контроллер, средства программирования и визуализации. Недостатком такого решения является значительное время восстановления при сбоях и повреждениях. Перезагрузка операционной системы (ОС) и запуск прикладной задачи может занимать несколько минут. Переустановка и настройка ОС, драйверов оборудования и прикладных программ требует значительного времени и высокой квалификации обслуживающего персонала, тогда как системное программное обеспечение ПЛК расположено в постоянной памяти в адресном пространстве центрального процессора и всегда готово к работе. По включению питания, ПЛК готов взять на себя управление системой уже через несколько миллисекунд.

## 1.3. **Основные характеристики и параметры ПЛК**

В методическом пособии рассмотрены вопросы практического использования контроллеров для автоматизации в различных областях техники на примере контроллеров Mitsubishi серии MELSEC FX2N и FX0S, как типичных представителей недорогих и широко используемых контроллеров. MELSEC FX0S − наиболее простой и недорогой логический контроллер, включающий в себя все преимущества системы ПЛК в компактном корпусе. Такое устройство представляет собой экономичную, с точки зрения вложений, альтернативу стандартным контакторным и релейным решениям. В свою очередь, MELSEC FX2N имеет более мощный процессор и обладает большей функциональностью, а также возможностью построения более сложных систем (т.е. с возможностью подключения модулей расширения и с поддержкой сетевого взаимодействия).

Для наглядности в таблице 1.1 приведены данные двух типичных спецификаций контроллеров.

Таблица 1.1

Технические характеристики контроллеров

| | FX2N -16MR-UA1/UL | FX0S-30MR-DS |
|---|---|---|
| **Электрические параметры** | | |
| Питание | ~100…240 В | = 12…24 В |
| Кол-во входов-выходов | 16 | 30 |
| Потребляемая мощность | 30 Вт | 8 Вт |
| Пиковый ток при включении | макс. 40А < 5мс/ ~100 В<br>макс. 60А < 5мс/ 200 В | макс 60А, <1.5 мс ,=24 В |
| Защита от КЗ | внешними цепями | внешними цепями |
| Предохранитель | 3.15А | 3.15А |
| Допустимый провал питания | 10 мс | 5 мс |
| Ток источника питания внутренней шины 5V DC | 290 мА | шина отсутствует |
| Ток сервисного источника питания 24V DC | 460 мА, пульсации при макс нагрузке: ≤ ±5% | нет сервисного источника |
| **Входы:** | | |
| Количество | 8 | 16 |
| Входной ток и напряжение (макс) | 4,7 мА / ~100 В /50Гц<br>6,2 мА/ ~110 В /60Гц | =24В / 8,5 мА |

Продолжение таблицы 1.1

| | | | |
|---|---|---|---|
| Ток переключения ВЫКЛ→ ВКЛ / ВКЛ → ВЫКЛ | | 80В 3.8мА / 30В 1.7мА мин ток лог1/ макс ток лог0 | >18В >4,5мА / <4В <1,5мА мин ток лог1/ макс ток лог0 |
| Быстродействие | | 25 мс | 0,1…15 мс , регулируемое |
| Входное сопротивление (импеданс) | | 21 кОм / 50 Гц 18 кОм / 60 Гц | |
| Гальваноразвязка | | опторазвязка между входами и питанием | |
| **Выходы:** | | | |
| Количество | | 8 | 14 |
| Тип выхода | | транзистор | реле |
| Уровень коммутируемого напряжения (макс) | | < ~240 В < =30 В | 5…30 В |
| Макс. выходной ток | - на канал | 0,5 А | 2 А |
| | - на группу | 0,8 А | 8 А |
| Коммутируемая мощность | индуктивная нагрузка | 12 Вт (0,5А / =24 В) | 80 ВА, ~120/ 240 В |
| | активная нагрузка | 1,5 Вт (0.0625А / =24 В) | 100 Вт (1.17 А / ~85 В 0.4 А / ~250 В) |
| Минимальная нагрузка | | | 5 мА , при < =24 В |
| Ток утечки | | <0.1 мА/ = 30 В | |
| Быстродействие | | <0,2 мс | 10 мс |
| Гальваноразвязка | | Реле | реле |
| Срок службы контактов (число коммутаций) | | бесконтактная коммутация | 3.000.000 при 20 ВА 1.000.000 при 35 ВА 200.000 при 80 ВА |
| **Механические параметры** | | | |
| Масса | | 0,35 кг | 0,5 кг |
| Размеры (Ш х В х Г) , мм | | 100 х 90 х 75 | 90 х 90 х 80 |
| **Условия эксплуатации** | | | |
| Диапазон рабочих температур | | 0 … +55˚ С | 0 … +55˚ С |
| Допустимая влажность воздуха | | 35…85 % (без конденсата) | 35…85 % (без конденсата) |
| Виброустой-чивость (2 часа в 3-х направлениях) | на винтах | 10..55 Гц / 0,5 мм / Макс 2g | 10..55 Гц / 0,5 мм / Макс 1g |
| | на DIN рейке | 10..55Гц / 0,5 мм / Макс 0,5g | 10..55 Гц / 0,5 мм / Макс 0,5g |
| Ударопрочность (3 цикла в 3-х направлениях) | | 10g | 10g |

| | | |
|---|---|---|
| Помехоустойчивость от генератора помех | 1000 Vpp, 1мс, при частоте 30..100 Гц | 1000 Vpp, 1мс @ 30..100 Гц |

Продолжение таблицы 1.1

| | | |
|---|---|---|
| Напряжение пробоя изоляции | ~1500 В, 1 мин<br>=500 В , 1 мин | ~1500 В, 1 мин<br>=500 В, 1 мин |
| Сопротивление изоляции | 5 МОм, U= 500В | 5 МОм, U= 500В |
| Заземление | Класс 3 | Класс 3 |
| Высота местности | До 2000 м | До 2000 м |
| Класс оборудования | II | II |
| Класс защиты | IP20 | IP20 |
| Степень загрязнения окр. среды | 2 | 2 |
| Окружающая среда | избегать сред, содержащих коррозийные газы и электропроводящую пыль и мусор<br>недопустимо воздействие активной среды, минимальное воздействие пыли | |
| **Программные параметры** | | |
| I/O (адресное пространство) | 256 | 128 (+4 опционально) |
| Диапазон адресов | макс 248 входов (X0-X367)<br>макс 248выходов(Y0-Y367) | макс 84 входа (X1-X123)<br>макс 64 выхода (Y0-Y77) |
| Память программы | 2000 шагов – EEPROM<br>+опциональные носители | 800 шагов – EEPROM |
| Быстродействие | 0,1..0,7мкс/лог.инструкцию | 0,55…1мкс/лог. инструкцию |
| **Операнды** | | |
| Количество инструкций (команд) | Базовых: 27<br>Прикладных: 128 | Базовых: 29<br>Прикладных: 85 |
| Внутреннее реле | 3072 | 512 |
| Спец. реле | 256 | 256 |
| Step-реле | 1000 | 74 |
| Таймеры | 256 | 84 |
| Задание установок внешними потенциометрами | 2 потенциометра | 1 потенциометр |
| Счетчики | 235 | 18 |
| Входы быстрого счета импульсов | 1-фазн. счет: 6 вх. до 60кГц<br>2-фазн. счет: 2 вх. до 30кГц | 7 вх. до 7 кГц<br>3 вх. до 14 кГц |

| | | |
|---|---|---|
| Часы реального времени | год/месяц/день/часы/ мин./сек./день недели | год/месяц/день/часы/ мин./сек./день недели |
| Регистры данных | 8000 | 8000 |

Окончание таблицы 1.1

| | | |
|---|---|---|
| Файловые регистры | Макс 7000 (всего ≤8000) | Макс 7000 (всего ≤8000) |
| Индексные регистры | 16 | 2 |
| Спец. регистры | 256 | 256 |
| Указатели | 128 | 64 |
| Допустимое число вложений в программе | 8 | 8 |
| Входы прерываний | 6 | 4 |
| **Дополнительно** | | |
| Масштабируемость (расширяемость) | да | нет |
| Число функциональных блоков в системе | 8 (ограничено адресным пространством I/O и током внутр. =5В шиной питания) | не поддерживает расширение |
| Поддержка работы по сети | используя сетевые модули расширения | не поддерживает сетевое взаимодействие |
| Встроенный интерфейс | RS-485 | RS-485 |

Многие из приведенных параметров очевидны и не возникает каких-либо вопросов по их расшифровке и оценке, некоторые были рассмотрены ранее или еще будут рассмотрены в рамках данного пособия, однако разберем некоторые из них более подробно.

## 1.4 **Питание**

### 1.4.1 Внешний источник питания

Так как работа ПЛК подразумевается в промышленных условиях, где то и дело приходится сталкиваться с резкими перепадами напряжения питающей сети, он должен быть нечувствительным к колебаниям напряжения. Так, допускается некоторое отклонение напряжения питания от номинальной величины (при питании переменным током ~100-240В +10%,-15% , 50/60 Гц ±10% или при питании постоянным током =24В +20%,-15%), а также полное кратковременное исчезновение (провал) питания на несколько миллисекунд. В это время приоритетно отрабатывается процедура защиты информации, и происходит пересылка содержимого регистров ЦП в энергонезависимую память. В некоторых ПЛК предусмотрена возможность автоматического пуска при восстановлении питания (либо с самого начала программы, либо с прерванной команды с восстановлением ситуации).

Для защиты контроллера при подключении питания необходимо использовать предохранитель, значение которого указано в спецификации. Смотри также раздел 3.4 «Подключение источника питания».

### 1.4.2 Пиковый ток при включении

Пиковый ток при включении – максимальный ток, который может быть необходим контроллеру во время пуска. Таким образом, необходимо выбирать источник питания достаточной мощности.

### 1.4.3 Внутренний источник питания

Внутренний (сервисный) источник питания постоянного тока =24 В предназначен для питания внешних устройств – датчиков и специальных модулей расширения, что делает удобным их подключение, однако следует учитывать значения потребляемого тока этими устройствами. (смотри раздел 5 «Расчет энергопотребления»). Для примера, сервисный источник питания постоянного тока контроллера FX2N 24 В, 460 мА; пульсации при максимальной нагрузке: ≤ ±5%.

### 1.4.4 Внутренняя шина

Шина предназначена для питания подключаемых модулей расширения ). В контроллерах, не поддерживающих расширение, она отсутствует.

## 1.5 **Входы и выходы ПЛК**

### 1.5.1 Дискретные входы

Один дискретный вход ПЛК способен принимать высокий или низкий уровень электрического сигнала, описываемый двумя состояниями – включено или выключено. На уровне программы – это один бит информации: 1 или 0, соответственно значение логической (булевой) переменной «ИСТИНА» или «ЛОЖЬ».

Кнопки, выключатели, контакты реле, датчики обнаружения предметов и множество приборов с выходом типа «реле» или «открытый коллектор» непосредственно могут быть подключены к дискретным входам ПЛК.

Некоторые первичные приборы систем промышленной автоматики имеют более 2-х состояний. Для их подключения используют несколько дискретных входов. Например, автоматические электронные весы способны контролировать пороги допуска. Они имеют 2 выхода – «меньше нормы» и «больше нормы». Вес объекта определяется двумя битами информации: 01 – «меньше», 00 – «норма», 01 – «больше», 11 – «неисправность прибора». Используя n отдельных входов можно закодировать $2^n$ состояний. Как правило, в прикладной программе ПЛК соответствующие биты объединяют в отдельную «дискретную» переменную.

Системное программное обеспечение ПЛК включает драйвер, автоматически считывающий физические значения входов в оперативную память. Благодаря этому, прикладному программисту нет необходимости разбираться с внутренним устройством контроллера. С точки зрения прикладного программиста дискретные входы это наборы бит, доступные для чтения.

Все дискретные входы контроллеров рассчитаны на прием стандартных сигналов согласно спецификации конкретного контроллера. Устройство входа включает индивидуальный светодиодный индикатор, гальваническую развязку и, как правило, защиту от ошибочного подключения. Так, для контроллера MELSEC FX0S (смотрите таблицу 2.1) уровень входного сигнала более 18 В считается логической единицей, а менее 4 В – логическим нулем. Типовое максимальное значение тока одного дискретного входа (при входном напряжении 24В) составляет около 10мА (8.5мА).

У многих контроллеров светодиодные индикаторы включены до гальванической развязки, что позволяет проводить диагностику работы внешних цепей, даже не включая контроллер.

Каждый дискретный вход имеет аналоговый фильтр «срезающий» высокочастотные помехи и верхние гармоники спектра входного сигнала. Частота среза фильтра согласована с программным быстродействием, определяющимся типовым временем рабочего цикла ПЛК. Длительность импульса, который можно надежно зафиксировать дискретным входом общего назначения, составляет 2…3 мс.

Все современные датчики, базирующиеся на разнообразных физических явлениях (емкостные, индуктивные, ультразвуковые, оптические и т.д.), как правило, поставляются со встроенными первичными преобразователями и не требуют дополнительного согласования при подключении к дискретным входам ПЛК.

Не смотря на внешнюю простоту дискретного входа, его схемотехническое решение и элементная база постоянно совершенствуются.

### 1.5.2 Аналоговые входы

Поскольку ПЛК является цифровой вычислительной машиной, аналоговые входные сигналы обязательно подвергаются аналого-цифровому преобразованию (АЦП). В результате образуется дискретная переменная определенной разрядности. Как правило, в ПЛК применяются 8…12 разрядные преобразователи, т.е. учитываются только 8…12 старших разрядов преобразованного аналогового сигнала. АЦП более высокой разрядности не оправдывают себя, в первую очередь из-за высокого уровня индустриальных помех, характерных для условий работы контроллеров.

Для аналоговых входов наиболее распространены стандартные диапазоны постоянного напряжения -10..+10В и 0..+10В. Для токовых входов это 0..20мА и 4..20мА. Для достижения хороших результатов измерений решающую роль играет качество выполнения монтажа внешних аналоговых цепей.

ПЛК может обладать встроенными аналоговыми входами или использовать дополнительно подключаемые модули расширения или адаптеры.

Особые классы аналоговых входов представляют входы, предназначенные для подключения термометров сопротивления и термопар. Здесь требуется применение специальных технических решений (трех-точечное включение, источники образцового тока, схемы компенсации холодного спая, схемы линеаризации и т.д.).

Практически все модули аналогового ввода являются многоканальными. Входной коммутатор подключает вход АЦП к необходимому входу модуля. Управление коммутатором и АЦП выполняет драйвер системного программного обеспечения ПЛК. Прикладной программист работает с готовыми значениями аналоговых величин в памяти контроллера аналогично дискретным входам.

### 1.5.3 Специальные входы

Стандартные дискретные входы ПЛК способны удовлетворить абсолютное большинство потребностей систем промышленной автоматики. Несоответствие физических значений напряжений и токов датчиков решается применением нормирующих преобразователей или заменой нестандартных датчиков. Здесь изготовление специализированных входов не оправдано.

Необходимость применения специализированных входов возникает в случаях, когда непосредственная обработка некоторого сигнала программно затруднена. Достаточно часто первичный сигнал содержит избыточную информацию, а программная фильтрация сложна или требует много времени.

Наиболее часто ПЛК оснащаются специализированными счетными входами для измерения длительности, фиксации фронтов и подсчета импульсов.

Например, при измерении положения и скорости вращения вала очень распространены устройства, формирующие определенное количество импульсов за один оборот – квадратурные шифраторы. Частота следования импульсов может достигать нескольких мегагерц. Даже если процессор ПЛК обладает достаточным быстродействием, непосредственный подсчет импульсов в пользовательской программе будет весьма расточительным по времени. Здесь желательно иметь специализированный аппаратный входной блок, способный провести первичную обработку и сформировать, необходимые для прикладной задачи, величины.

Вторым распространенным специализированным типом входов являются входы способные очень быстро запускать заданные пользовательские задачи с прерыванием выполнения основной программы.

### 1.5.4 Дискретные выходы

Один дискретный выход ПЛК способен коммутировать один электрический сигнал. Также как и дискретный вход, с точки зрения программы это один бит информации, принимающий состояния «ИСТИНА» или «ЛОЖЬ».

Простейший дискретный выход ПЛК выполняется в виде контактов реле. Такой выход достаточно удобен в применении и прост. Однако он обладает характерными недостатками реле – ограниченный ресурс, низкое быстродействие, разрушение контактов при работе на индуктивную нагрузку. Альтернативным решением дискретного выхода является электронный силовой элемент, который выполняется по бесконтактной схеме (транзистор – для нагрузки постоянного тока, симистор – для нагрузки переменного тока). Схема ключа, как правило, содержит индивидуальную светодиодную индикацию, гальваническую развязку и элементы защиты от ошибочного включения и короткого замыкания нагрузки.

Практика эксплуатации доказала нецелесообразность сосредоточения в корпусе ПЛК большого числа силовых коммутирующих элементов. Оптимальным решением является установка силовых коммутирующих приборов максимально близко к нагрузке. В результате, сокращается длина силовых монтажных соединений, снижается стоимость монтажа, упрощается обслуживание, уменьшается уровень электромагнитных помех. Поэтому наиболее широким спросом пользуются дискретные выходы средней мощности (до 1А, 24В).

Светодиодные индикаторы включения выходов питаются от ПЛК. Это упрощает отладку программы управления без подключения оборудования.

Благодаря применению специальных узлов защиты, дискретный выход контроллера обладает очень высокой надежностью.

### 1.5.5 Аналоговые выходы

Как и в случае с аналоговыми входами для управления аналоговыми сигналами ПЛК может обладать встроенными аналоговыми выходами или использовать дополнительно подключаемые модули расширения или адаптеры .

**Примечание**

− Практика показывает, что аналоговые выходы менее надежные, чем дискретные, поэтому при конфигурации системы полезно иметь некоторый запас по количеству аналоговых выходов (то же можно сказать и о входах).

− Также полезно иметь запас по количеству дискретных входов/выходов, т.к. часто бывает необходимо подключить уже на стадии эксплуатации дополнительное оборудование, не прибегая к покупке дополнительных модулей расширения. Таким образом, разумно иметь примерно 20% запас свободных каналов.

## 1.6 **Время реакции – быстродействие ПЛК**

Быстродействие – это основополагающий фактор, влияющий на выбор ПЛК. Примером быстродействия может служить то время, которое требуется человеку для осмысления ответа на поставленный вопрос или для принятия решения. Так и ПЛК необходимо некоторое время для обработки текущего состояния. Для некоторых приложений этот параметр может не являться критическим, и выбор того или иного значения быстродействия должен сопоставляться с реальными требованиями приложения. Так и для контроллера время реакции зависит, как от числа опрашиваемых входов/выходов, так и от производительности самого процессора. Давайте попробуем проанализировать эту ситуацию. Факторы, влияющие на время реакции контроллера, изображены в виде схемы на рисунке 1.4.

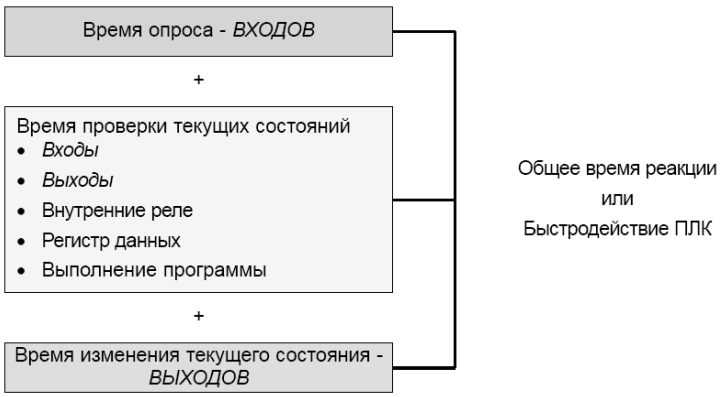



Рис. 1.4. Время реакции или быстродействие ПЛК

Теперь, когда мы получили представление о том, как работает контроллер, давайте посмотрим, что в действительности происходит в ПЛК при обработке текущих состояний входов, и как это влияет на время срабатывания выходов. Разберем три типичных случая переключения входов ПЛК, приведенных на рисунке 1.5.

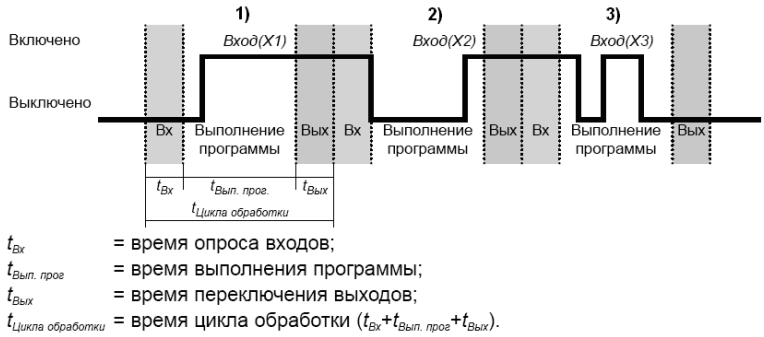



Рис. 1.5. Пример переключения входов ПЛК

1) ПЛК не зафиксировал переключение входа (X1). Это произошло потому, что переключение произошло после обработки контроллером состояний входов. А значит, контроллер сможет выполнить операцию, соответствующую входу (X1) только на следующим цикле опроса.

2) ПЛК не зафиксировал переключение входа (X2). Следовательно, если контроллер должен был выключить выход (Y1) при одновременном

включении входа (X1) и входа (X2), то это событие не было обработано контроллером и операция не выполнена.

3) ПЛК не зафиксировал переключение входа (X3). И, возможно, не сможет обработать изменение состояния данного входа (X3) никогда.

Есть три варианта решения этой проблемы:

1) Использовать контроллер с более высоким быстродействием;

2) Воспользоваться функцией задержки времени «выкл». Эта функция увеличит длительность входного сигнала (смотри рисунок 1.6)

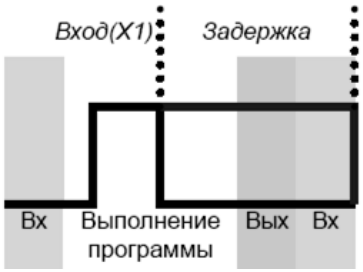



Рис. 1.6. Задержка времени включения

3) Использовать функцию обработки прерываний (рисунок 1.7). То есть, как только вход включен, то независимо от того этапа, на котором в настоящий момент находится программа ПЛК, немедленно останавливает выполнение основной программы и выполняет подпрограмму прерывания.

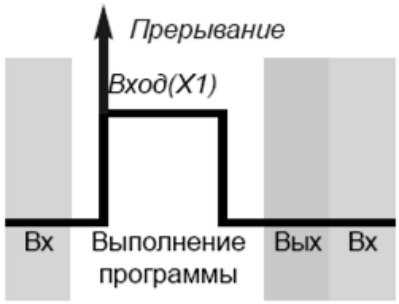



Рис. 1.7. Использование прерывания

Таким образом, из рисунка 1.8 следует: можно считать, что оптимальным временем продолжительности импульса будет являться время прохода одного цикла программы.

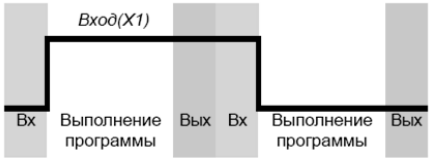
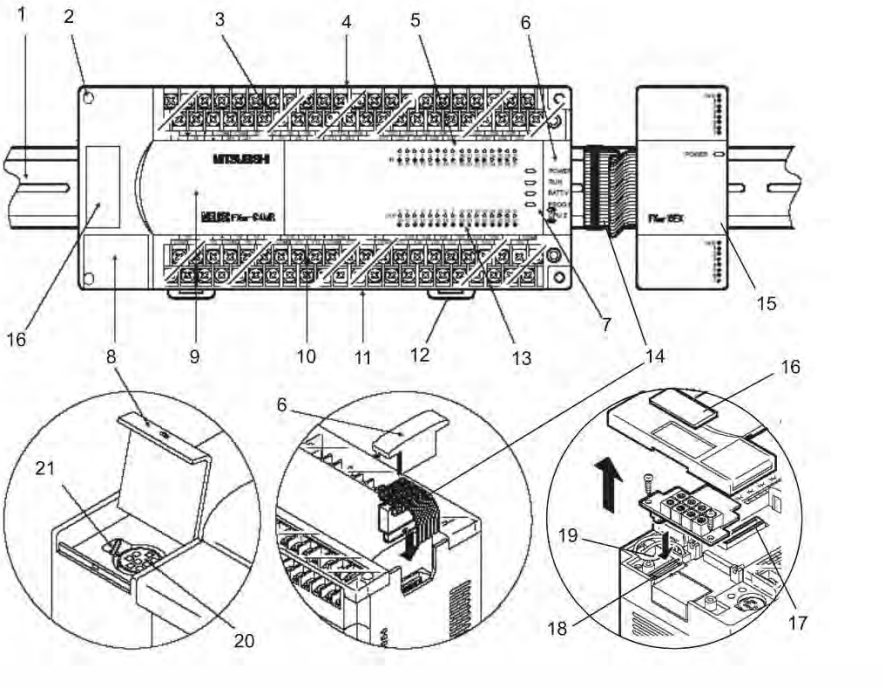



Рис. 1.8. Временная диаграмма цикла работы ПЛК

В технических характеристиках ПЛК приводится типовое время рабочего цикла. (в нашем случае в таблице 1.1 приведены составляющие величины рабочего цикла – быстродействие входов, выходов и время исполнения логических инструкций программы). При его измерении пользовательская программа должна содержать 1К (1024) простых логических команд (на языке списка инструкций (IL) по стандарту МЭК 1131-3). Сегодня ПЛК имеют типовое значение времени рабочего цикла, измеряемое единицами миллисекунд и менее. События, требующие быстрой реакции, выделяются в отдельные задачи, приоритетность и период выполнения которых можно изменять.

## 1.7 **Установка и подключение ПЛК**

Рассмотрим основные конструктивные элементы контроллера на примере Mitsubishi MELSEC FX2N (рисунок 1.9).




1 - DIN-рейка 35мм (1.38 дюйма);
2 - Установочные отверстия;
3 - Входные клеммы;
4 - Заглушка входных клемм;
5 - Индикатор состояния входов;
6 - Заглушка шины расширения;
7 - Индикаторы состояния;
8 - Заглушка порта программирования;
9 - Передняя панель;
10 - Выходные клеммы;
11 - Заглушка выходных клемм;
12 - Фиксатор DIN-рейки;
13 - Индикатор состояния выходов;
14 - Шлейф для подключения модуля расширения;
15 - Модуль расширения;
16 - Заглушка разъема адаптера расширения;
17 - Порт для установки кассеты памяти;
18 - Разъем для адаптера расширения;
19 - Батарея для сохранения данных при отсутствии питания;
20 - Порт программирования;
21 - Переключатель ВКЛ/ВЫКЛ (RUN/STOP);

Рис. 1.9. Общий вид контроллера MELSEC FX2N

Обычно блоки промышленных контроллеров устанавливаются на 35 мм (1,38 дюйма) DIN-рейку. Крепление блоков на DIN-рейке осуществляется при помощи специальной защелки. Возможно также крепление на панель с помощью винтов в отверстия.

Подключение модулей расширения, специальных функциональных модулей к базовому процессорному модулю выполняется при помощи

стандартного плоского шлейфного кабеля либо специальных выводных корпусных контактов.

Как правило, на промышленных объектах контроллеры устанавливаются в специальные шкафы управления, в которых предусмотрены стандартные разъемы и DIN рейки для крепления основных и периферийных блоков и модулей, что позволяет легко и удобно компоновать целые системы управления. На рисунке 1.10 приведен пример расположения контроллера в шкафу управления.

Однорядное расположение

Двухрядное расположение с использованием дополнительного шлейфного кабеля

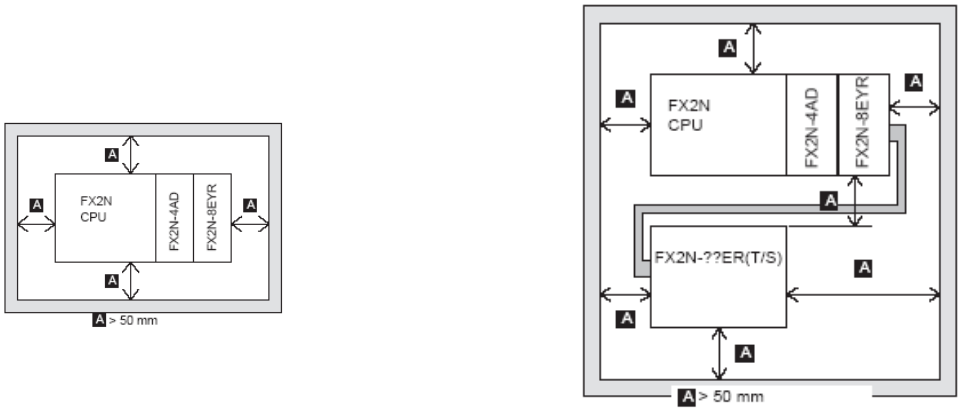



Рис. 1.10. Пример расположения контроллеров в шкафу управления

При установке контроллера необходимо обратить особое внимание на соблюдение всех условий эксплуатации, приведенных в техническом паспорте контроллера. В основном различают три типа факторов воздействия агрессивной среды:

▪ *Физические и механические факторы*:

К ним относятся диапазон рабочих температур, атмосферное давление (высота над уровнем моря), влажность, вибрация и удары. Расположение в непосредственной близости от нагревателей, химических реакторов, доменных печей, а также тяжелые климатические условия могут негативно отразиться на работоспособности элементов устройств управления (откуда вытекает необходимость создания систем естественной или принудительной вентиляции ПЛК). Высокий уровень влажности (более 80%) вызывает конденсацию паров и ускоряет коррозию. Уровень влажности менее 35% способствует возникновению электростатических зарядов, вызывающих случайные срабатывания логических схем. При установке устройств рядом с источниками вибрации и ударов сварные, контактные и другие соединения подвергаются опасным воздействиям. Так, рассматриваемые нами контроллеры способны работать при температурах 0…+55˚ С и влажности 35…85 %, выдерживают вибрацию с частотой от 10 до 55Гц с амплитудой 0,5 мм и максимальным ускорением от 0,5 до 2g в течение 2-х часов в каждом

направлении X, Y, Z. Также данные контроллеры выдерживают ударную нагрузку с ускорением 10g (3 цикла в каждом направлении X, Y, Z).

▪ *Химические факторы:*

К ним относят вызывающие коррозию газы ($Cl_2$, $H_2S$, $SO_2$), углеводородные пары, металлическая (литейный, сталеплавильный цеха) или минеральная (цементный завод) пыль. Являющаяся действием этих факторов коррозия поражает контакты и микросхемы. Для ее предотвращения чаще всего покрывают лаком платы с печатными схемами и устанавливают фильтры, препятствующие попаданию пыли или агрессивных газов. Иногда ПЛК делают полностью герметичными.

▪ *Электрические факторы:*

Обычно энергия электрической промышленной помехи может достигать 100 мкДж. Энергия переключения схемы ТТЛ при напряжении 5 В, токе 2 мА и продолжительности переключения 10 нс равна $10^{-4}$ мкДж, т.е в $10^6$ раз меньше. Отсюда следует, что уровень помех должен быть снижен на 120 дБ. Основными источниками помех являются: термоЭДС (эффект Пельтье) в несколько милливольт, разность потенциалов в зоне контакта металлов с различной химической активностью, электростатические и электромагнитные влияния, вызываемые включением индуктивностей и емкостей (расположение вблизи трансформаторов, сварочных агрегатов и др.). Помехи от источников двух первых типов могут внести погрешность в результаты измерений аналоговых величин низкого уровня или вызвать коррозию элементов; для защиты от помех от источников двух последних типов необходимо соответствующее исполнение входов-выходов, например обеспечение эффективной гальванической развязки (оптрон, реле, разделительный трансформатор).

Приведенные в таблице 1.1 данные указывают, что рассматриваемые контроллеры имеют гальваническую развязку входов и выходов и обладают помехозащищенностью от импульсного напряжения амплитудой 1000 В от генератора помех длительностью 1 мс при 30…100 Гц (1000 Vpp, 1 мс при частоте 30…100 Гц).

## 1.8 **Общие рекомендации по электробезопасности**

Как и все электротехнические устройства ПЛК должен соответствовать определенным требованиям электробезопасности. Обычно качество изоляции тестируется между всеми рабочими точками устройства, а также корпусом и землей (методика проведения испытаний по МЭК 61131-2 Программируемые контроллеры. Общие технические требования и методы испытаний).

Используется проводник определенного сечения (обычно не менее 2 мм$^2$). Сопротивление заземления должно быть менее 100 Ом (класс 3). Заземляющий проводник должен быть присоединен к тому же контуру заземления, что и силовые цепи. Эти требования также указываются приведенным в предыдущем пункте стандартом.

Условный номер, присвоенный группе оборудования, для которой применяются определенные средства, используемые для обеспечения защиты от поражения электрическим током при нормальной эксплуатации и в условиях единичного дефекта функционирования установленного оборудования.

Таким образом, оборудование класса II (см. табл.1.1) — оборудование с прочным и по существу непрерывным корпусом из изоляционного материала, который окружает все проводящие части, за исключением небольших частей типа фабричных марок, винтов и заклепок, отделенных от частей с опасным напряжением изоляцией, эквивалентной, по крайней мере, усиленной изоляции. Подробнее смотри стандарт МЭК 61131-2.

Код IP показывает степень защиты, которую обеспечивает оболочка (корпус). Расшифровку конкретного кода можно найти в справочнике или в ГОСТ 14254-96. Так, IP20 (смотри таблицу 1.1) расшифровывается следующим образом:

2 – ПЛК защищен от проникновения внешних твердых предметов диаметром >12,5 мм (это же означает, что человек (оператор) не может пальцами достать до опасных частей ПЛК).

0 – ПЛК не имеет защиты от вредного воздействия в результате проникновения воды.

Условный номер, присвоенный для оценки изолирующих качеств воздушных зазоров и частей поверхности изоляции с целью установления величины загрязнения микросреды:

1) **степень загрязнения 1:** Отсутствие загрязнений или наличие только сухих, непроводящих загрязнений. Загрязнения не существенны;

2) **степень загрязнения 2:** Обычно имеют место только непроводящие загрязнения. Иногда может ожидаться временная проводимость, вызванная конденсацией влаги;

3) **степень загрязнения 3:** Имеют место проводящие загрязнения. Сухие непроводящие загрязнения могут стать проводящими из-за конденсации влаги. Все входящие и выходящие кабели присоединяются к ПЛК к специальному разъему «под винт». Кабельный ввод имеет крышку для защиты от коротких замыканий и нарушения контакта.

**ПРЕДУПРЕЖДЕНИЕ**

1. Не используйте для входных и выходных сигналов один и тот же провод.
2. Не используйте для входных и выходных сигналов жилы одного и того же многожильного кабеля.
3. Не располагайте сигнальные кабели вблизи силовых. Низковольтные кабели должны быть отделены или изолированы от силовых.
4. При значительной длине проводников входных/выходных сигналов необходимо учитывать потерю напряжения и шумовые помехи.

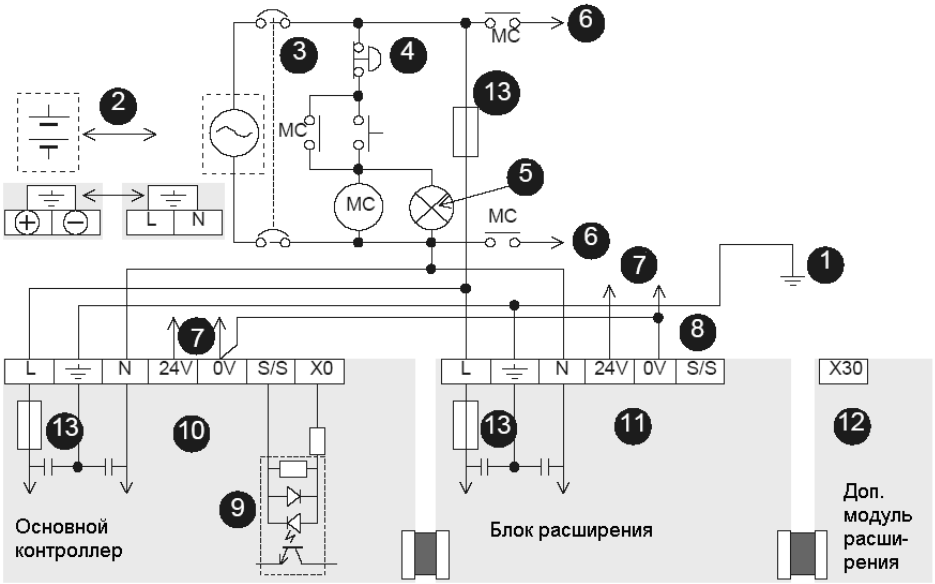
При использовании источника питания постоянного тока необходимо соединить «+» клемму источника питания с «+» клеммой контроллера, а «−» клемму источника питания с «−» клеммой контролера. Ни какое другое подключение недопустимо.

При использовании источника питания переменного тока:

− линия *фазы* должна быть связана с клеммой *L* а линия *0* с клеммой *N*. Во избежание поражения электротоком не соединяйте линию *фазы L* с клеммой *N*.

− подключение заземления обязательно. Сопротивление заземления должно быть R < 100 Ом

Пример подключения источника питания показан на рисунке 1.11




1 - Заземление: класс 3;
2 - Источник питания;
3 - Защитное устройство;
4 - Аварийный стоп;
5 - Индикатор наличия питания;
6 - Питание нагрузки;
7 - Встроенный источник питания;
8 - Переключатель типа коммутации входов;
9 - Оптопара;
10 - Базовый блок;
11 - Блок расширения;
12 - Модуль расширения (нпр. аналоговый вход);
13 - Предохранитель.(на 3 А).

Рис. 1.11. Пример подключения источника питания к ПЛК

**ВАЖНЫЕ ЗАМЕЧАНИЯ:**

**Встроенный источник питания:**

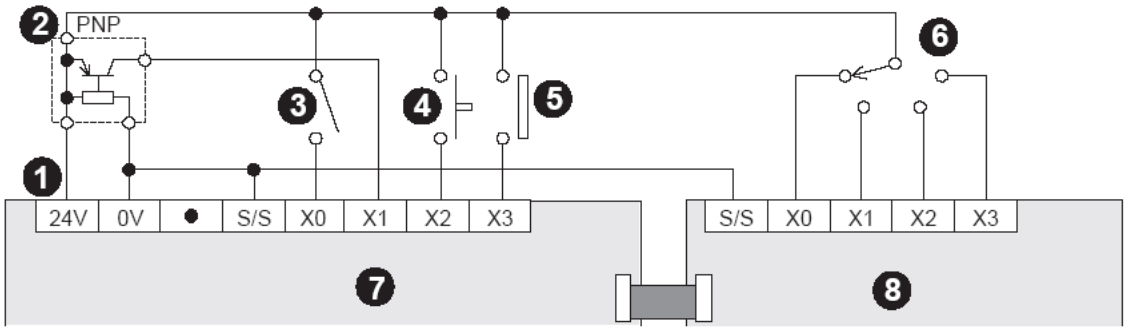
− Если в системе используется встроенный источник питания как базового блока, так и блока расширения, то клеммы «0» должны быть соединены.

− В то же время НЕ СОЕДИНЯЙТЕ клеммы «24 В» базового блока и блока расширения между собой;

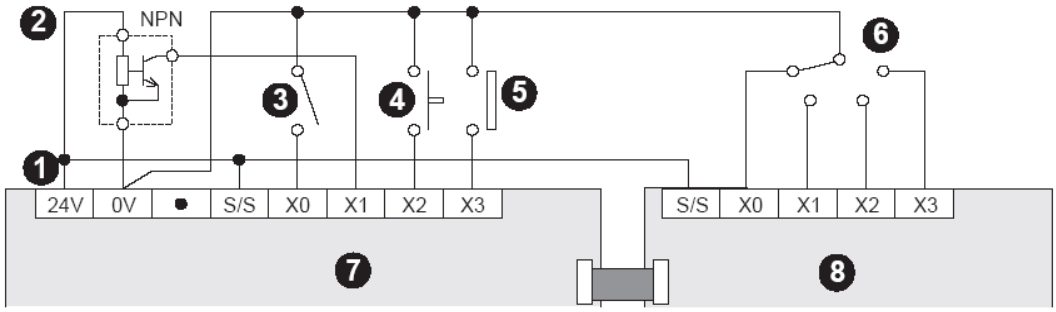
− НИКОГДА не подключайте внешний источник питания к клеммам «24 В».

## 1.9 **Подключение входов выходов**

В зависимости от внутренней реализации входов ПЛК может быть два типа подключения: коммутация общим плюсом (source) , либо общим минусом (sink). Пример подключения показан на рисунке 1.12. Многие контроллеры поддерживают оба типа подключения. В нашем примере выбор типа подключения осуществляется клеммой «S/S».




коммутация общим плюсом (source)




коммутация общим минусом (sink)

1 - Источник питания постоянного тока;
2 - Датчик приближения PNP (NPN);
3 - Переключатель;
4 - Кнопка;
5 - Контакт;
6 - Поворотный переключатель;
7 - Базовый блок;
8 - Модуль расширения.

Рис. 1.12. Пример подключения входов контроллера

Особенности подключения входов ПЛК к внешним устройствам (например, датчикам) в зависимости от способа их питания показаны на рисунках 1.13...1.15.

Подключение датчика, использующего встроенный в ПЛК источник питания

Подключение датчика, использующего независимый источник питания

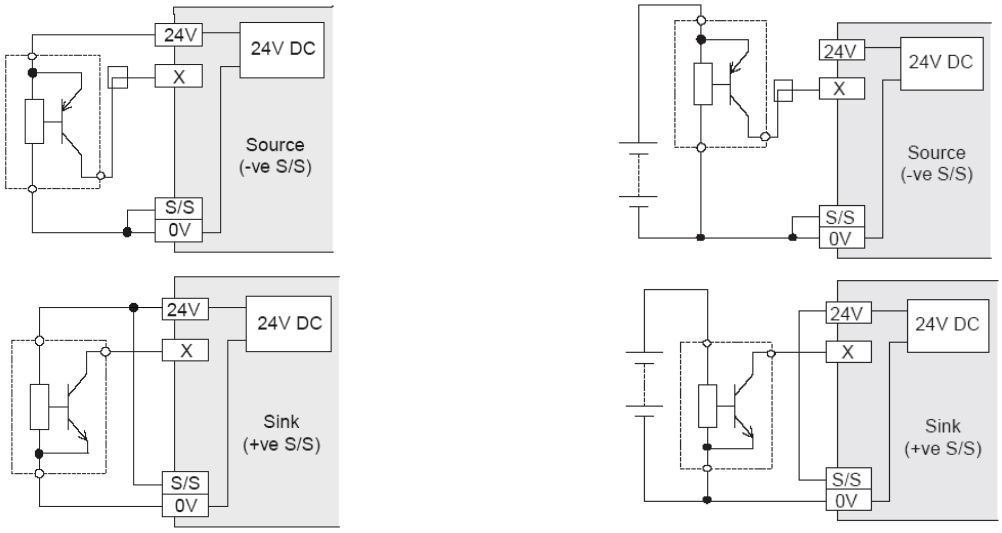



Рис. 1.13. Схема подключения датчика к входу ПЛК в зависимости от способа питания датчика и способа коммутации входов ПЛК

Последовательное подключение диодов и входов

На рисунке 3.6 показано последовательное подключение диода и входа ПЛК. В этом случае при нажатии на кнопку диод загорается. Важно учитывать, что максимальное падение напряжения на диоде – 4 В. Последовательно может быть подключено не более 2 светодиодов.

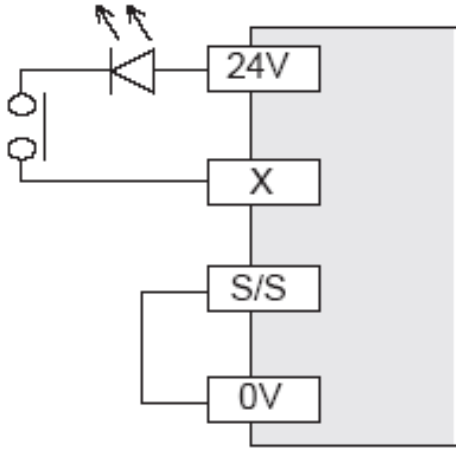



Рис. 1.14. Последовательное подключение диода

Также учитывается, что максимальный входной ток для входа ПЛК (например, 8,5 мА), а потребление тока диодом примерно 2,1мА. При этом необходимо обеспечить корректное срабатывание входа, т.е. значение тока

логической «1» и «0». (например, для FX0S: >4,5 мА и <1,5 мА для «1» и «0» соответственно, смотри таблицу 2.1).

Параллельное подключение резисторов и входов. Параллельное подключение светодиода

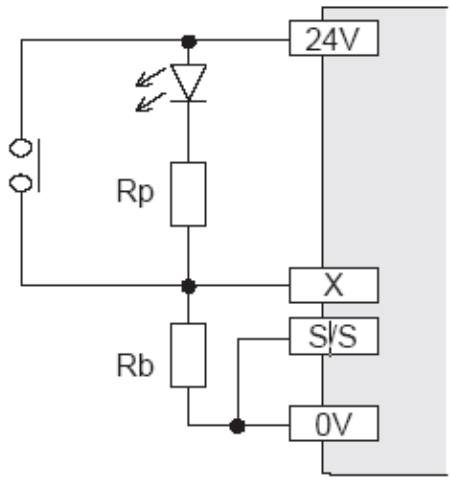



Рис. 1.15. Параллельное подключение диода

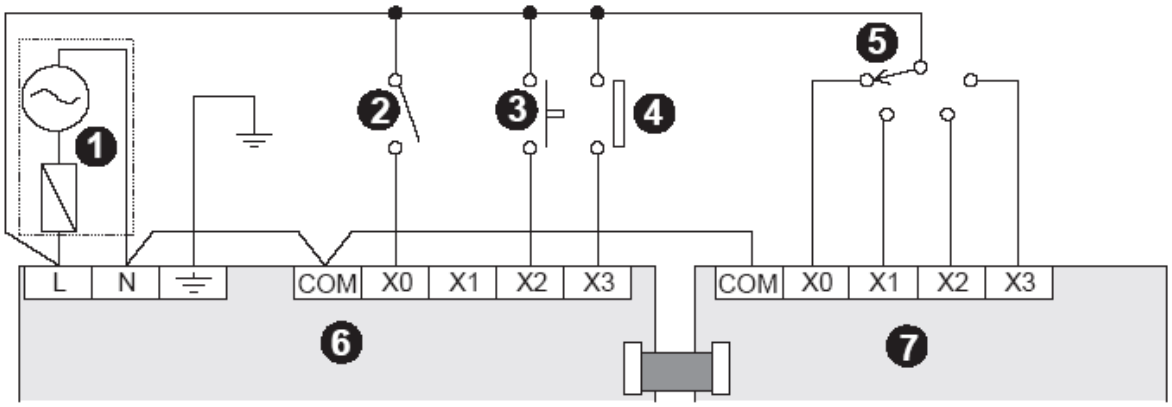
При таком подключении (рисунок 3.7) светодиод при нажатой кнопке погасает. Для сохранения уровня входного тока, поступающего на входы ПЛК, и сохранения работоспособности светодиода необходимо параллельно подключить сопротивление Rp: для FX2N = 15 кОм. Если сопротивление Rp меньше установленного значения, то добавляется сопротивление Rb. Для определения значения Rb смотрите уравнение (3.1).

Ток переключения из состояния логической «1» в «0» для FX2N = 1,5 мА (смотри таблицу 2.1). Если ток больше установленного значения, то ставится дополнительный резистор *Rb* (подтяжка к нулю). Таким образом, вход будет корректно обрабатывать сигнал. Для определения значения Rb смотрите уравнение (3.2).

$$Rb \le \frac{4Rp}{15 - Rp} \qquad (1)$$

$$Rb \le \frac{6}{I - 1.5} \qquad (2)$$

Для закрепления вышеизложенного материала на рисунке 1.16 приведен пример подключения входов контроллера и модуля расширения, работающего от источника переменного тока, к дискретным элементам СУ.



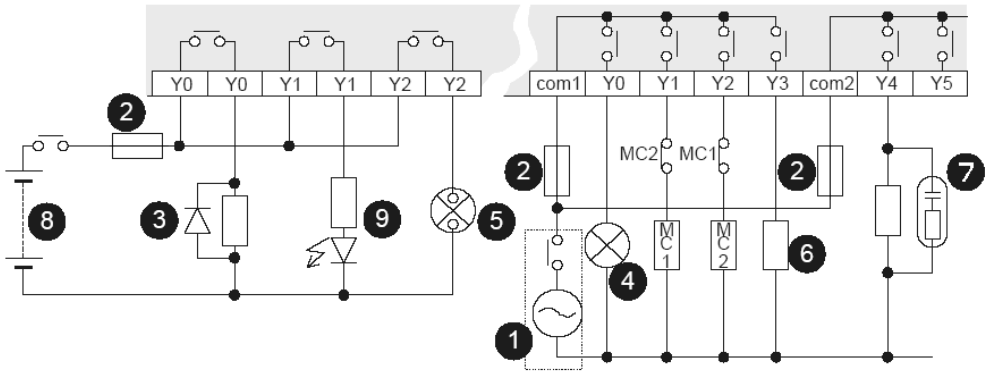

1 - Источник питания переменного тока;
2 - Переключатель;
3 - Кнопка;
4 - Контакт;
5 - Поворотный переключатель;
6 - Базовый блок;
7 - Модуль расширения.

Рис.1.16. Пример подключения входов контроллера

При подключении нагрузки к выходам ПЛК необходимо учитывать следующие параметры ПЛК :

- номинальное напряжение (резистивная нагрузка), В
- номинальный ток / N точек (резистивная нагрузка), А
- максимальная индуктивная нагрузка, ВА
- максимальная активная нагрузка, Вт
- минимальная нагрузка, (необходимая нагрузка при определенном малом напряжении для релейных выходов)
- время срабатывания, мс
- ток утечки (для транзисторных выходов)
- изоляция контура

В зависимости от типа выходов ПЛК подключение осуществляется разными способами. На рисунках 1.17…1.19 приведены соответственно схемы подключения для релейных, транзисторных и симисторных выходов. Следует обратить внимание, что транзисторные выходы рассчитаны на нагрузку постоянного тока, симисторные – на нагрузку переменного тока, релейные выходы бывают для обоих типов нагрузки (обычно различаются расположением выходных клемм, смотри рисунок 1.17).




Подключение выхода к нагрузке с источником питания постоянного тока

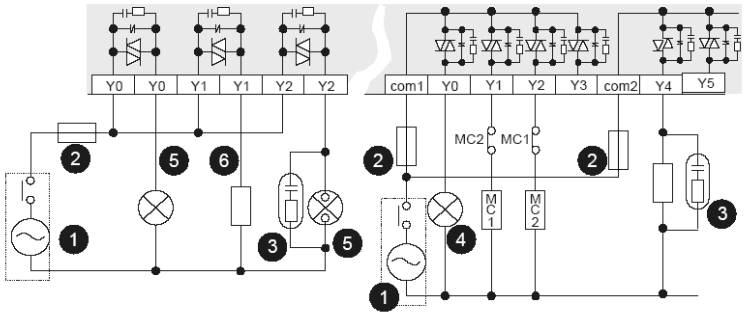
1 – Источник питания переменного тока;
2 – Предохранитель (1-2 А на каждую группу из 4-х входов для защиты выходных
цепей ПЛК);
3 – Токоограничивающий диод (продлевает срок службы реле).
4 – Лампа накаливания;

Подключение выхода к нагрузке с источником питания переменного тока

5 – Неоновая лампа;
6 – Резистор;
7 – Цепь помехозащиты (уменьшает шумы при индуктивной нагрузке ~) (C = 0.1 мкФ, R = 100…120 Ом)
8 – Источник питания постоянного тока;
9 – Светодиод.

Рис. 1.17. Схема подключения релейных выходов

Схема подключения симисторных выходов ПЛК показана на рисунке 3.10.



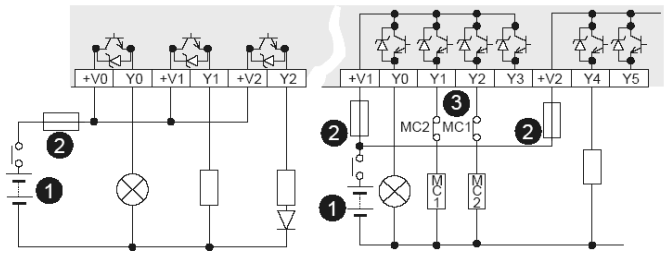

1 – Источник питания переменного тока;
2 – Предохранитель;
3 – Цепочка помехозащиты;
4 – Лампа накаливания;
5 – Неоновая лампа;
6 – Резистор.

Рис. 1.18. Схема подключения симисторных выходов

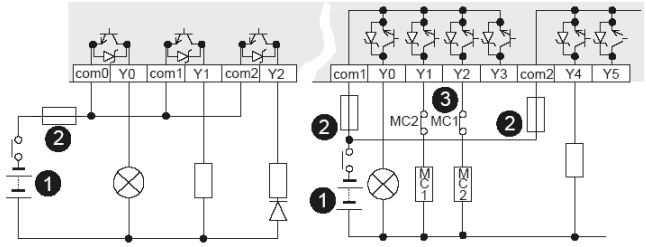
Подключение транзисторных выходов также может производиться двумя способами:

- коммутация общим плюсом (Source);
- коммутация общим минусом (Sink).

Пример схем таких подключений выходов ПЛК к различного рода нагрузке показан на рисунке 1.19.




коммутация общим плюсом (Source)




коммутация общим минусом (Sink)

1 – Источник питания постоянного тока;
2 – Предохранитель;
3 – Внешняя механическая блокировка логически исключающих друг друга выходов (например, вращение двигателя вперед/назад)

Рис. 1.19. Схема подключения транзисторных выходов

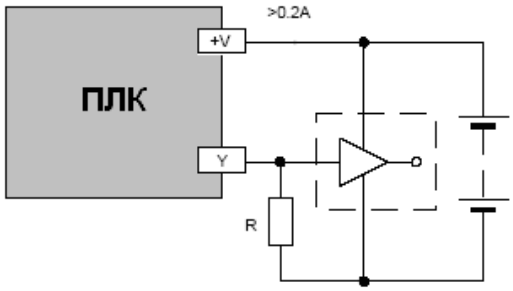



Рис. 1.20. Подключение добавочного резистора к выходу ПЛК

Для достижения оптимального времени срабатывания транзисторного выхода рекомендуется использовать в выходных внешних цепях добавочный

резистор R, как показано на рисунке 1.20. Значение добавочного резистора выбирается таким, чтобы значение выходного тока было не менее 0,2 А.

Пример схемы подключения источника питания к контроллеру FX0S, а также подключение его входов и выходов к технологическому оборудованию (датчикам, исполнительным механизмам) приведен на рисунке 1.21. В данном примере контроллер управляет движением конвейеров и толкателя, а также подает управляющие сигналы на СУ роботом.

Контроллер питается от источника постоянного тока 24 В, входы ПЛК подключены к датчикам по схеме коммутации общим плюсом (source), релейные выходы подключены к нагрузке постоянного тока. Для каждой группы из 4-х выходов имеется отдельный предохранитель.

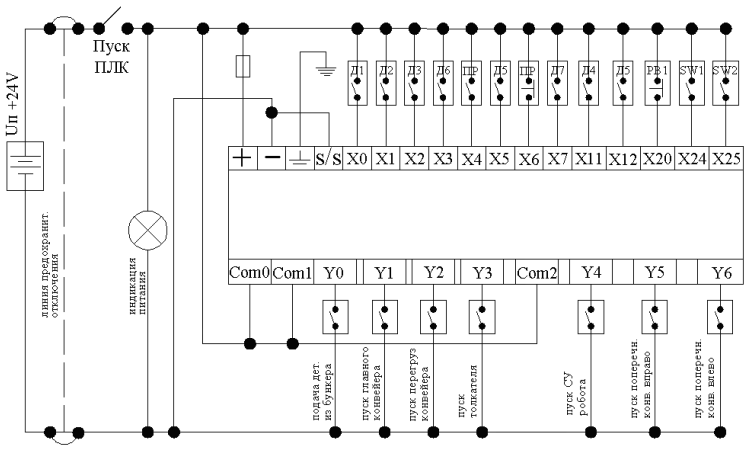
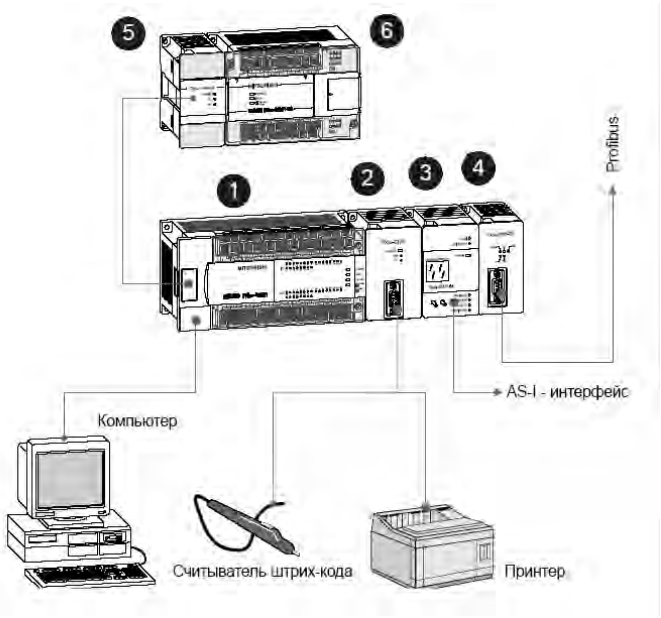



Рис. 1.21. Схема подключения ПЛК FX0S к источнику питания и технологическому оборудованию

## 1.10 **Конфигурация системы**

Часто возникают задачи, требующие расширения функциональности контроллера и построения более мощной и гибкой системы управления с набором специальных возможностей, но на базе одного основного блока. Быстро нарастить и расширить возможности контроллера позволяют дополнительно подключаемые блоки и модули (модули расширения). Пример такой системы показан на рисунке 1.22.




1 - Основной блок;
2 - Интерфейсный модуль RS232C;
3 - Интерфейсный модуль AS-I;
4 - Интерфейсный модуль PROFIBUS DP;
5 - Интерфейсный модуль RS485;
6 - Ведомый основной блок (slave).

Рис. 1.22. Пример многофункциональной расширенной системы управления на базе одного основного блока

Если возникает необходимость задействовать в проекте большее количество входов/выходов, то ПЛК может быть расширен с помощью соответствующих устройств. На примере контроллеров Mitsubishi серии MELSEC это может быть либо компактный блок расширения, модуль расширения, либо адаптеры (рисунок 1.23).

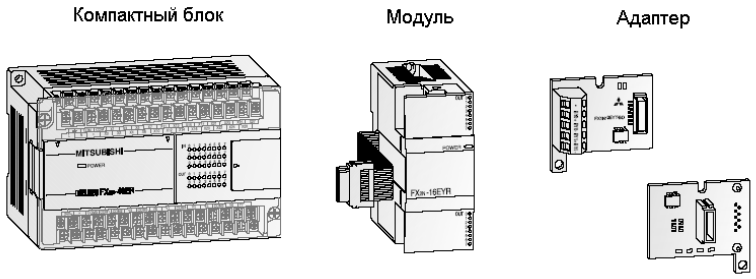


Рис. 1.23. Средства наращивания количества входов-выходов

Компактные блоки имеют 32 или 48 входов/выходов (релейные и транзисторные модификации) со светодиодной индикацией состояния входов/выходов и встроенным источником питания (что очень важно при расчете энергопотребления – см. ниже).

Модули расширения имеют 4, 8 или 16 входов/выходов (релейные и транзисторные модификации), также оснащены светодиодной индикацией, однако не имеют собственного источника питания.

Адаптеры расширения на 4 входа и 2 выхода устанавливаются непосредственно в базовые модули контроллеров и не вызывают изменения их габаритных размеров. Адаптеры не занимают адресного пространства контроллера, для входов и выходов предусмотрены отдельные адреса, имеют светодиодную индикацию состояния.

Аналоговые модули могут оснащаться разным числом входов/выходов (В/В), которые преобразуют цифровые величины контроллеров в аналоговые сигналы (ЦАП), и наоборот, аналоговые сигналы (например, с датчиков температуры или давления) – в цифровые величины для дальнейшей обработки контроллером (АЦП). Наряду с этим имеются готовые модули со встроенной термопарой и ПИД-регулятором для управления температурой, имеющие определенное число каналов, управляемых раздельно. Модули аналоговых В/В показаны на рисунке 1.24.

Аналоговый адаптер расширения представляет собой ЦАП и АЦП с одним входом/выходом и может выдавать сигнал как по току, так и по напряжению. Устанавливается в слот расширения базового модуля, дополнительное питание не требуется.


Рис. 1.24. Средства для обработки аналоговых величин

## 1.11 **Основы программирования ПЛК**

Теперь, когда мы имеем представление об основных принципах работы ПЛК, как он обрабатывает входы, выходы, и как выполняется программа, мы готовы начать писать программу. Но для начала вспомним о том, как работает реле, и что реле из себя представляет, так как целью использования контролера и является физическое замещение реле.

В принципе реле – это выключатель, подобно тем выключателям, которыми мы пользуемся повседневно, включая-выключая свет в комнатах. Только, если дома мы используем физическую силу для включения-выключения, то промышленное реле использует электромагнитное поле. При подачи напряжение на катушку возбуждается магнитное поле – это магнитное поле притягивает контакты реле, следствием чего является соединение двух контактов. В сущности, основное назначение реле – дать возможность току протекать между двумя контактами.

Давайте рассмотрим простой пример, в котором необходимо включить электродвигатель, при нажатии на выключатель вход (X1).

В результате мы получим устройство (рисунок 1.25), которое можно условно разделить на три основные части:

- выключатель вход (X1);
- реле;
- электродвигатель.

Всякий раз, когда выключатель вход (X1) соединяет контакты (включается), цепь замыкается и ток, создавая электромагнитное поле в реле, переключает контакты реле, т.е. пускает электродвигатель.

В данном примере использовано типичное индустриальное реле постоянного тока для управления включением-выключением устройств. Пока вход (X1) открыт, ток не может течь через катушку реле и электродвигатель не работает. Но как только вход (X1) закрыт, ток создает электромагнитное поле, которое заставляет контакты реле соединиться. В результате ток, протекающий через контакты реле, вращает электродвигатель.

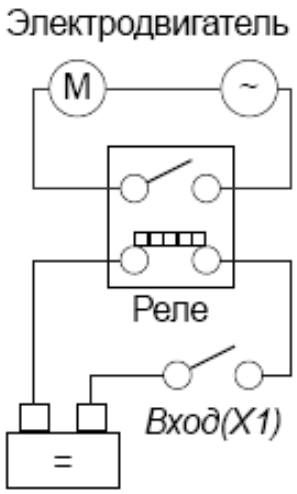


Рис. 1.25. Пример использования реле

Попробуем заменить реле контроллером. Конечно, данный пример не является показателем эффективности использования ПЛК в категории цена-производительность. Но он позволит понять, в каких случаях и для чего можно эффективно использовать контроллер. Первое, что необходимо осознать, каким образом мы можем объяснить ПЛК то, какую задачу он должен выполнить.

Одним из базовых языков программирования ПЛК остается язык *Релейно-контактных схем*. (наиболее удобный для понимания сути в нашем примере)

Шаг первый:

Необходимо переопределить все составляющие оборудования, которые мы используем, в символы, понятные для контроллера, так как ПЛК ничего не знает о существовании таких вещей, как выключатель, реле, электродвигатель и т.д. ПЛК может работать с символами вход, выход. Для контроллера совершенно не важно, что из себя физически представляет вход или выход. контроллер обрабатывает только текущее состояние входа (включено-выключено). Все остальные действия выполняются последовательно и только в строгом соответствии с алгоритмом, заложенным в контроллер программой – т.е. Вами.

Шаг второй:

Сначала заменяем источник питания. Для языка *Релейно-контактных схем* - этим символом будет являться две параллельные прямые с левой и правой стороны диаграммы. Можно считать, что левая линия является «+», а правая линия «–».

Шаг третий:

Присваиваем символы входам. В нашем примере имеем два входа:

*Вход (X1)* ⊣ ⊢ – нормально открытый контакт;

*Вход (X2)* ⊣/⊢ – нормально закрытый контакт.

Шаг четвертый:

присваиваем символ выходам.

*Выход (Y1)* —(Y1)— – символ катушки реле.

В результате мы получили программу, которая может быть выполнена ПЛК (рисунок 1.26).

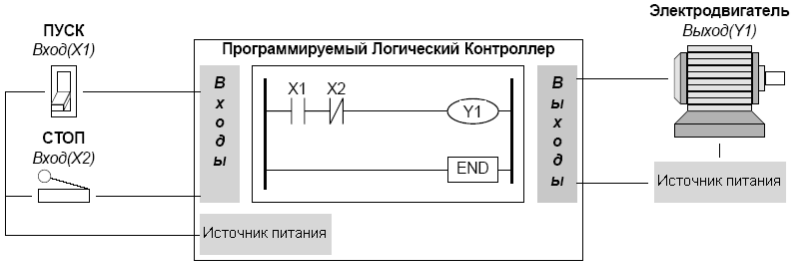


Рис. 1.26. Пример программы для ПЛК

## 1.12 **Языки программирования, пакеты ПО**

В 1993 году Международной Электротехнической Комиссией (МЭК) был опубликован стандарт МЭК 61131-3 (IEC 61131-3). Этот международный стандарт входит в группу стандартов МЭК 1131, которые охватывают различные аспекты использования программируемых логических контроллеров. 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 ждать его адаптации не хватило сил: функции поддержки и внедрения стандарта на рынке взяла на себя независимая организация PLCOpen, состоящая из производителей и пользователей программного обеспечения (ПО), ориентированного на МЭК 61131-3. В результате деятельности PLCOpen на рынке ПО появилась серия сертифицированных средств программирования ПЛК,– средств, которые достаточно широко и небезуспешно внедряются в промышленности.

Список инструментальных программных систем, реализующих стандарт МЭК 61131-3, превышает два десятка (таблица 1.2).

Таблица 1.2

Примеры инструментальных систем, реализующих МЭК 61131-3

| Название программного пакета | Производитель |
|---|---|
| CoDeSys | Smart Software Solutions, Германия |
| ACCON-ProSys | Deltalogic, Германия |
| OpenDK | Infoteam Software, Германия |
| PUMA | KEBA, Австрия |
| SUCOsoft S340 | Klonker-Moeller, Германия |
| NAIS CONTROL | Matsushita AC, Германия |
| PDS7 | Philips, Нидерланды |
| SELECONTROL | Selection Lyss, Швейцария |
| Soft Control | Softing, Германия |
| ISaGRAF | CJ International, Франция |

## 1.13 **Организация PLCopen и уровни совместимости**

Общую координацию деятельности производителей и пользователей «61131»-систем (инструментальные системы, реализующие стандарт МЭК 61131-3 на пять языков программирования контроллеров) осуществляет независимая международная Ассоциация PLCopen. Она занимается популяризацией и информационной поддержкой стандарта МЭК 61131-3 с целью его использования в промышленных системах контроля и управления.

В задачи PLCopen не входит поддержка разработки универсального инструмента программирования для любого типа контроллеров. Основное направление деятельности ассоциации – поддержка некоторого множества языков программирования, использование которых позволит пользователям различных контроллеров обмениваться своими наработками.

Одна из важнейших задач PLCopen – выработка системы и принципов сертификации программных продуктов на предмет их соответствия стандарту МЭК 61131-3. PLCopen определяет три уровня совместимости инструментальных систем (рисунок 1.27):

- Базовый уровень (Base Level)
- Уровень переносимости функций (Portability Level)
- Уровень переносимости приложений (Application Level)

Базовый уровень предполагает, что системы должны быть совместимы на некотором подмножестве базовых компонентов, определяемых стандартом (типы переменных, языковые конструкции и т. п.). Это первый этап, который должны пройти разработчики контроллеров с возможностями подключения к «61131»-системе.

Следующий уровень совместимости определен как уровень переносимости функций и функциональных блоков между различными системами. Для этой цели был введен специальный формат файла обмена.

Последний, третий уровень, определяет степень совместимости и переносимости на уровне завершенных приложений.

Хорошо проработаны и ведутся процедуры сертификации инструментальных «61131»-систем по первым двум уровням. Спецификация по третьему уровню совместимости (а это мечта любого системного интегратора) пока разрабатывается.

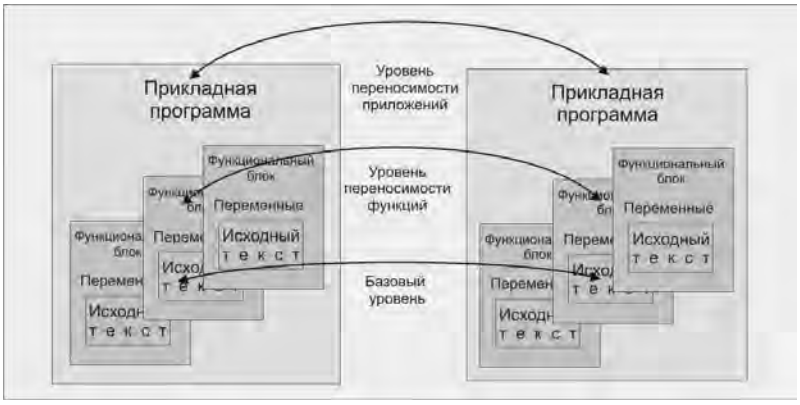


Рис. 1.27. Уровни совместимости «61131» – систем

## 1.14 **Классификация языков по стандарту МЭК 61131**-3

Стандарт МЭК 61131-3 определяет языки для программируемых контроллеров таким образом, что части прикладной программы могут быть запрограммированы на любом языке и скомпонованы в единую исполняемую программу. 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.

Опыт показывает, что выбор используемых языков чаще всего определяется личными предпочтениями пользователей и мало связан с автоматизируемым технологическим процессом. Действительно, представленные в стандарте языки в большинстве случаев взаимозаменяемы. Это значит, что при разном уровне подготовленности в области чистого программирования пользователи могут создавать программы равной функциональности.

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.

### 1.14.1 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 (рисунок 1.28)

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

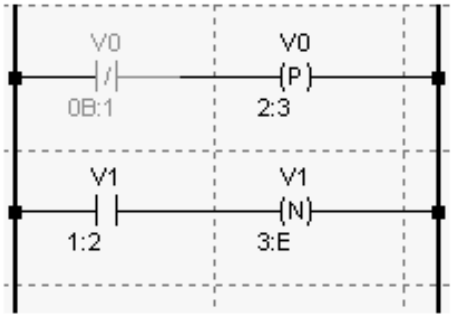


Рисунок 1.28 – Пример кода программы на языке
релейно-контактных схем

### 1.14.2 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), а также описывать последовательно-параллельные задачи в понятной и наглядной форме (рисунок 1.29).

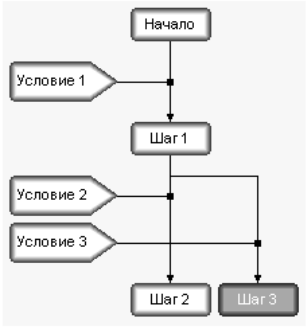



Рисунок 1.29 – Пример кода программы на языке
последовательных функциональных схем

Строго говоря, SFC не является языком программирования. Это средство проектирования прикладного программного обеспечения, которое всегда 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 – все 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 максимально 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 ранних этапах проектирования прикладного ПО позволяет снять многочисленные непонимания между заказчиком, проектировщиком ПО и программистом.

### 1.14.3 Язык функциональных блоков (FBD)

Язык функциональных блоков (Function Block Diagrams) позволяет создать программную единицу практически любой сложности на основе стандартных кирпичиков (арифметические, тригонометрические, логические блоки, ПИД-регуляторы, блоки, описывающие некоторые законы управления, мультиплексоры и т.д.). На рисунке 1.30 показан пример типичных элементов. Это языковое средство использует технологию инкапсуляции алгоритмов обработки данных и законов регулирования. Все программирование сводится к «склеиванию» готовых компонентов. В результате получается максимально наглядная и хорошо контролируемая программная единица.

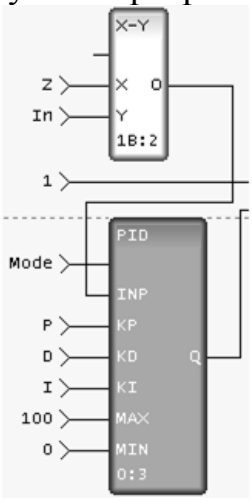



Рисунок 1.30 – Пример кода программы на языке функциональных блоков

### 1.14.4 Язык списка инструкций (IL)

В «достандартные» времена (до 1993 года) практически каждый программируемый контроллер сопровождался своим Ассемблером. Выросли целые поколения программистов, ориентированных на определенные кланы микропроцессоров. Освоение новой техники сталкивалось с проблемой освоения очередного языка программирования под новый кристалл. Отдельные мнемонические конструкции Ассемблеров были похожи, но о каком-либо стандарте не было и речи.

Появление языка инструкций (Instruction List) в наборе стандартных языков – это унификация интерфейса языка программирования низкого уровня, неориентированного на какую-либо микропроцессорную архитектуру. У языка IL есть очень важное качество: на его основе можно создавать оптимальные по быстродействию программные единицы. Пример участка кода на языке инструкций приведен на рисунке 1.31.

```
ADD VAR_000 2.6
LT VAR_000 VAR_001
JMPC label1
GT VAR_001 20
JMPC label2
LD 278
label1:
CAL FUNCTION1(VAR_000, 3)
label2: ST VAR_001
```

Рисунок 1.31 – Пример кода программы на языке списка инструкций

### 1.14.5 Язык структурированного текста (ST)

Язык структурированного текста (Structured Text) относится к классу текстовых языков высокого уровня (пример кода программы на рисунке 1.32). 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 шагов и транзакций языка SFC. Кроме этого, он имеет «выходы» во все остальные языки, что делает его универсальным в применении разными категориями пользователей.

```
PROGRAM
    VAR_INPUT ARG_000 : REAL; END_VAR
    VAR_INOUT ARG_001 : REAL; END_VAR

WHILE ARG_000 > 2 DO ARG_000 = ARG_000-1;
ARG_001 = ARG_001 + 2;
END_WHILE;

END_PROGRAM
```

Рисунок 1.32 – Пример кода программы на языке структурированного текста

В многих инструментальных «61131»-системах существует возможность «смешивать» программы/процедуры, написанные на разных языках, а также вставлять кодовые последовательности из одного языка в коды, написанные на другом языке.

## 1.15 **Язык релейно-контактных схем (LD)**

Рассмотрим язык релейно-контактных схем, дающий наглядное представление о работе контроллера за счет того, что изначально контроллер создавался для замены реле. Таким образом, данный язык нагляден и понятен программисту, который только начинает знакомиться с программной логикой.

Структура команды на языке релейно-контактных схем показана на рисунке 1.33.

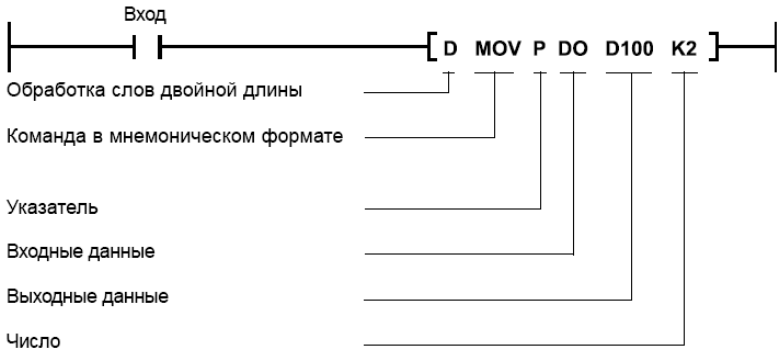
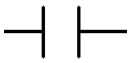



Рисунок 1.33 – Структура команды

### *1.15.1 Основные команды*

*Команда (LD) - нормально открытый контакт*



Прочитав этот сигнал, контроллер начинает постоянно проверять состояние входа, норме которого указан программистом. (Далее для простоты изложения будут указаны конкретные номера, они выбраны произвольно). И как только контролер определяет изменение состояния входа (X1), с выключено на включено, следует включение выхода (Y1). Данный символ может относиться не только к физическим входам контроллера, а также и к внутренним (вспомогательным) реле. Число и ограничения по использованию внутренних реле определяется только возможностями контроллера.

*Команда (LDI) - нормально закрытый контакт*

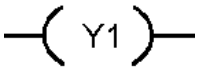
Прочитав этот сигнал, контроллер начинает постоянно проверять состояние входа (X2). И как только контроллер определяет изменение состояния входа (X2), с включено на выключено, следует включение выхода (Y1). Данный символ может относиться не только к физическим входам

контроллера, а также и к внутренним (вспомогательным) реле. Логика работы соответствует таблице 1.3.

Таблица 1.3 – Таблица логического состояния входов

| Логическое состояние | *LD* - нормально открытый контакт | *LDI* – нормально закрытый контакт |
|---|---|---|
| 0 | Ложь | Истина |
| 1 | Истина | Ложь |

*Команда (OUT) - инициализация* ***Выхода***




Прочитав эту команду, контроллер изменит состояние выхода (Y1), с выключено на включено. Так же, как и в случае с входами данный символ может относиться не только к физическим выходам контроллера, но и к внутренним (вспомогательным) реле. Число и ограничения по использованию внутренних реле определяется только возможностями контроллера. Логика работы соответствует таблице 1.4.

Таблица 1.4 – Таблица логического состояния выходов

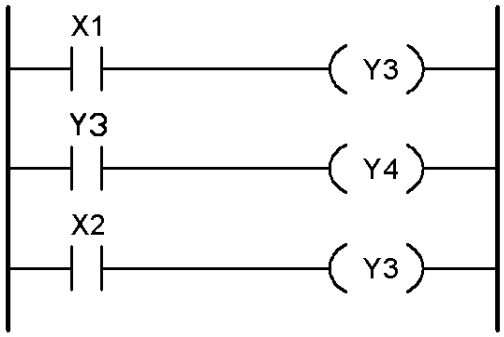
| Логическое состояние | *OUT* – Выход |
|---|---|
| 0 | **Ложь** |
| 1 | **Истина** |

**Примечание:**

Избегайте двойной записи выходов (double coil), так как это может привести к помехам при отработке программы.

Пример двойной записи приведен на рисунке 1.34а – ошибочная, б – верная запись:

а)




б)

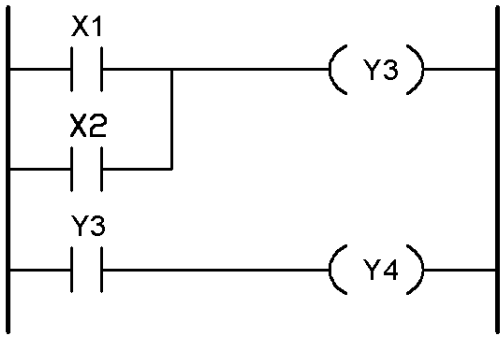



Рисунок 1.34 – Пример двойной записи выхода

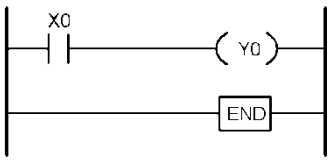
Исходим из того, что вход X1 включен (сигнал «1»), а вход X2 отключен (сигнал «0»).

Первая запись выхода Y3 активизируется включенным входом X1, в отображении процесса выходов Y3 включен, соответственно активизируется также выход Y4.

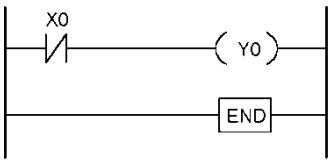
Эта программная последовательность имеет следствием то, что когда выключится X1, Y3 останется включенным, при условии, что был включен X2.

Для исправления такой ошибки, необходимо использовать оператор «ИЛИ». (см. далее)

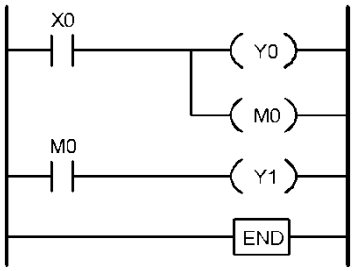
Отработка зависимости выходов от значения входов показана на рисунке 1.35:




Y0 будет в состоянии «истина», когда вход X0 будет включен («истина») , т.е. замкнут.




Y0 будет в состоянии «истина», когда вход X0 будет включен («истина») , т.е. замкнут.




Вместе с Y0 будет включено внутреннее реле M0 (см. далее), замыкание которого повлечет в свою очередь установление выхода Y1 в состояние 1 («истина»).

Рисунок 1.35 – Зависимость отработки выходов от значения входов

**Команда (AND) - логическое умножение**

Операция логического умножения. В языках программирования и языках запросов обозначается символами AND, И, & и другими способами. Результатом операции является «истина», если оба операнда принимают значение «истина», и «ложь» – в остальных случаях. Пример записи команды показан на рисунке 1.36.

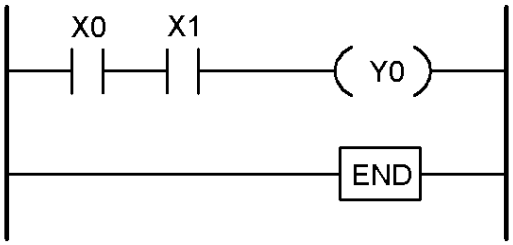
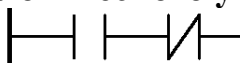



Рисунок 1.36 - Логическое умножение

**Команда (ANI) - отрицание логического умножения**



По аналогии с предыдущим операндом результатом операции является «истина», если X0 будет замкнут, и X1 – разомкнут, т.е. оба входа примут значение «истина», в остальных случаях Y0 будет «ложь». Пример записи команды показан на рисунке 1.37.

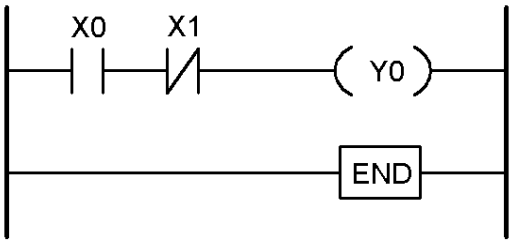
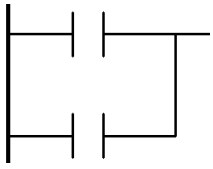



Рисунок 1.37 – Отрицание логического умножения

**Команда (OR) - логическое сложение**



Операция логического сложения. В языках программирования и языках запросов обозначается символами OR и другими способами. Результатом операции является «истина», если оба или один из операндов принимают значение «истина», и «ложь» – в остальных

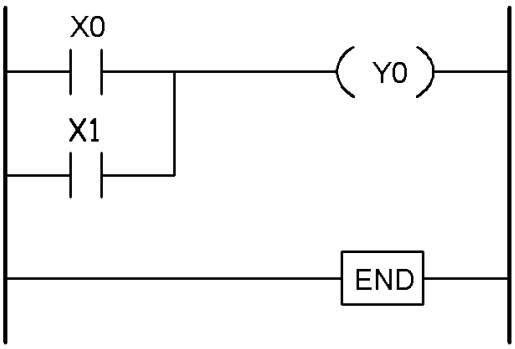
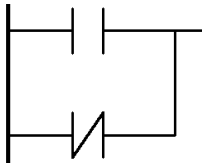



Рисунок 1.38 - Логическое сложение

случаях. Пример записи команды показан на рисунке 1.38.

**Команда (ORI) - отрицание логического сложения**



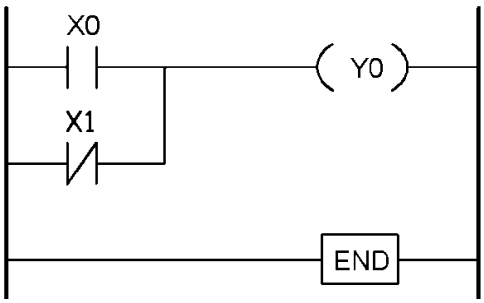
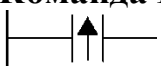



Рисунок 1.39 – Отрицание логического сложения

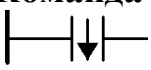
*Команды (LDP) и (LDF) – управление по фронтам входных сигналов*

При необходимости использовать фронты сигналов входов (передний или задний), сигнал входа будет иметь вид:

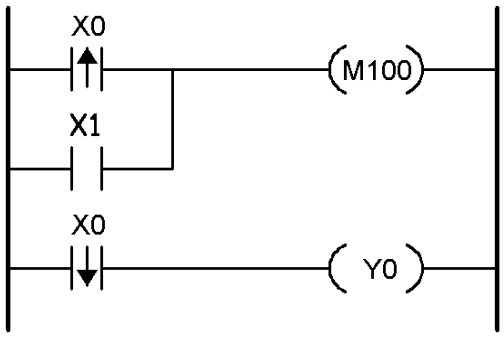
**Команда LDP (управление по переднему фронту)**



**Команда LDF (управление по заднему фронту)**



Пример использования управления по фронтам сигналов показан на рисунке 1.40.




Внутреннее реле M100 (меркер) включается на время включения X1 или при положительном фронте X0 (моменте его включения).

Выход Y0 включается при отрицательном фронте X0 (моменте его отключения).

Рисунок 1.40 – Управления по фронтам сигналов

*Команды SET(Установить)/RST(Сбросить)*

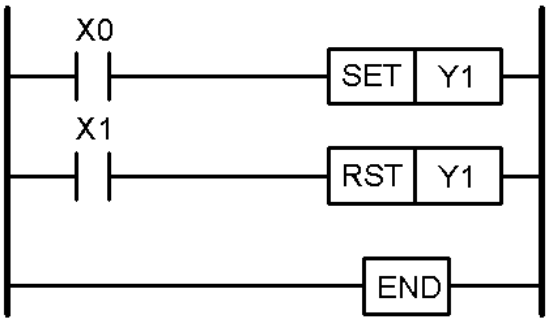



Рисунок 1.41 – Команды прямой установки

Пример, объясняющий функциональность команд прямой установки и сброса, показан на рисунке 1.41.

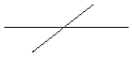
Состояние сигнала операнда с помощью «SET/RST» команд (включение/выключение) может устанавливаться непосредственно.

С помощью «SET/RST» могут устанавливаться в «1»/«0» (включаться/выключаться) соответствующие операнды, например: Y (выход) , M (внутреннее реле) или S (состояние шагов).

Также «RST» применяется для обнуления регистров и счетчиков.

**Примечание:** команда «RST» преобладает над командой «SET».

*Команда (INV) – Инверсия результата обработки*

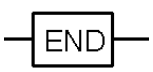


Инвертирует состояние сигнала результата стоящей впереди команды. Полученный согласно обработки сигнал «1», после инверсии становится «0», и соответственно наоборот «0» становится «1».

*Команда (NOP) – Пустая строка в программе*

Можно создать пустую строку без логических функций, которая позднее может быть использована для каких-либо команд, например, при окончательном изготовлении программы, при отладке оборудования. После успешного завершения отладки программы «NOP»-команды должны быть удалены, так как в противном случае они бесполезно удлиняют время цикла программы. «NOP» – команда может использоваться для создания паузы нужной длительности при отработке программы ПЛК.

*Команда (END) – конец программы*




Окончание программы ПЛК и переход к началу программы (шаг 0). Каждая программа ПЛК должна завершаться командой «END» . На этом месте

оканчивается обработка программы. Последующие области программы не принимаются во внимание. После обработки «END»–команды выполняется обработка выходов. Чтобы организовать участки программы для пошаговой проверки, можно вводить «END»–команду также внутри программы. Эта дополнительная «END»–команда должна после окончания проверки удаляться.

## 1.16 **Программирование внутреннего реле**

Давайте вернемся к предыдущему примеру и посмотрим, как контроллер обрабатывает состояния входов-выходов, и хранит полученные значения. Приведем вновь рисунок, упрощенно изображающий ПЛК для управления технологическим процессом и его программу (рисунок 1.42).

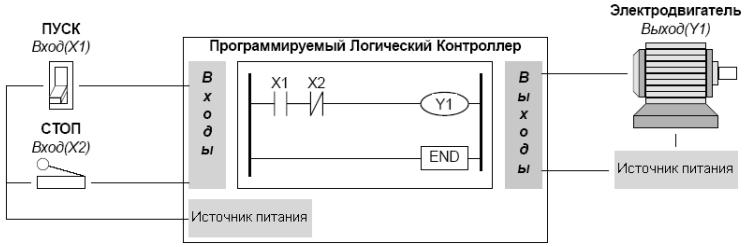



Рисунок 1.42 – Пример программы для ПЛК

Таблица 1.5а, б дает представление о том, как контроллер хранит полученные значения в регистрах:

Таблица 1.5 а

Пример состояние регистров входов и выходов ПЛК

| ***Регистр входов (X0…X15)*** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *X15* | *X14* | *X13* | *X12* | *X11* | *X10* | *X9* | *X8* | *X7* | *X6* | *X5* | *X4* | *X3* | *X2* | *X1* | *X0* |
| | | | | | | | | | | | | | 1 | 0 | |

| ***Регистр выходов (Y0…X15)*** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Y15* | *Y14* | *Y13* | *Y12* | *Y11* | *Y10* | *Y9* | *Y8* | *Y7* | *Y6* | *Y5* | *Y4* | *Y3* | *Y2* | *Y1* | *Y0* |
| | | | | | | | | | | | | | | 0 | |

**Регистр входов:**

− в регистре X1 находится значение «0» (бит «0»), то есть вход (X1) – выключен;

− в регистре X2 находится значение «1» (бит «1») следовательно, вход (X2) – включен.

**Регистр выходов:**

− в регистре Y1 находится значение «0» (Бит «0»), то есть выход (Y1) – выключен.

В действительности во всех остальных ячейках находится значение «0», но они не отображены, чтобы сосредоточиться на примере.

Рассмотрим то, как изменятся значения в регистрах после того, как включится выключатель – вход (X1). Значения регистров выходов приведены в таблице 1.5 б.

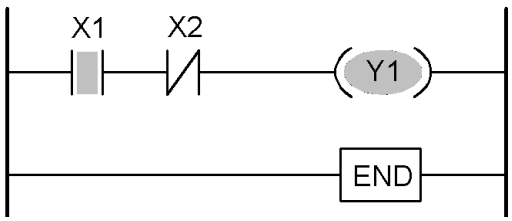



Рисунок 1.43 – Реакция выхода на изменение состояния входа

Таблица 1.5 б – Пример состояние регистров входов и выходов ПЛК

| ***Регистры входов (X0…X15)*** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *X15* | *X14* | *X13* | *X12* | *X11* | *X10* | *X9* | *X8* | *X7* | *X6* | *X5* | *X4* | *X3* | *X2* | *X1* | *X0* |
| | | | | | | | | | | | | | 1 | 1 | |

| ***Регистры выходов (Y0…X15)*** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Y15* | *Y14* | *Y13* | *Y12* | *Y11* | *Y10* | *Y9* | *Y8* | *Y7* | *Y6* | *Y5* | *Y4* | *Y3* | *Y2* | *Y1* | *Y0* |
| | | | | | | | | | | | | | | 1 | |

Согласно схемы после включения входа (X1) включился и выход (Y1) – электродвигатель заработал и технологический процесс пошёл… Рисунок 1.43 иллюстрирует это.

Таблица 1.6 описывает все возможные состояния для двух входов и одного выхода.

Таблица 1.6 – Таблица возможных состояний входов и выходов ПЛК

| ***Входы ( X1/X2 )*** | | ***Выход ( Y1 )*** | ***Значение в регистрах*** | | |
|---|---|---|---|---|---|
| *LD ( X1 )* | *LDI ( X2 )* | *OUT ( Y1)* | *LD ( X1 )* | *LDI ( X2 )* | *OUT ( Y1)* |
| Ложь | Истина | Ложь | 0 | 0 | 0 |
| Истина | Истина | Истина | 1 | 0 | 1 |
| Истина | Ложь | Ложь | 1 | 1 | 0 |
| Ложь | Ложь | Ложь | 0 | 1 | 0 |

Из таблицы 1.6 можем видеть, что контроллер выполнит операцию включения выхода (Y1) тогда, когда вход (X1) и вход (X2) будут в состоянии «истина».

Данный пример показывает (рисунок 1.44), как работает логика «AND» («И»), т.е. выход (Y1) изменит свое состояние «0» на «1» тогда и только тогда, когда вход (X1) и вход (X2) будут в состоянии «истина». Во всех остальных случаях выход (Y1) будет в состоянии «0», т.е. «выкл.».

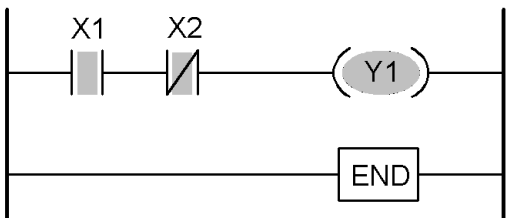



Рисунок 1.44 – Пример работы логики «И»

**Использование внутреннего реле (меркера) – контроль уровня.**

Предположим нам необходимо постоянно поддерживать определенный уровень в баке с водой (рисунок 1.46). Для решения необходимо:

- датчик верхнего уровня – вход (X2) (переходит в состояние «включено», когда уровень воды падает);
- датчик нижнего уровня – вход (X1) (переходит в состояние «включено», когда уровень воды падает);
- насос – выход (Y1);

К сожалению, использование логики для двух входов не даст желаемого результата, так как если мы будем включать насос, когда оба входа окажутся в состоянии «включено», то насос будет поднимать уровень только до нижнего датчика, или уровень воды никогда не опустится до уровня нижнего датчика. Т.е. начнет работать в очень жестком режиме постоянного включения-выключения.

Для обеспечения работы насоса в оптимальном режиме нам понадобится включение в задачу дополнительного элемента, который должен обеспечить:

- *Включение* насоса при достижении нижнего уровня;
- *Выключение* насоса при достижении верхнего уровня.

Для этого мы будем использовать внутренне реле контроллера M1, также называемое меркером, в результате чего получим схему, изображенную на рисунке 1.45.

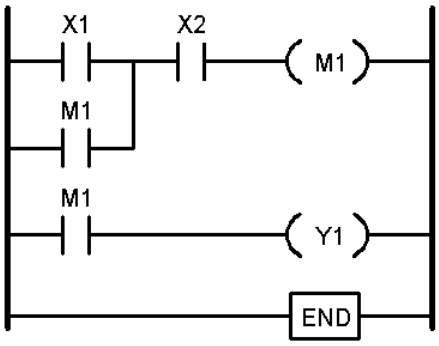



Рисунок 1.45 – Схема программы для включения насоса

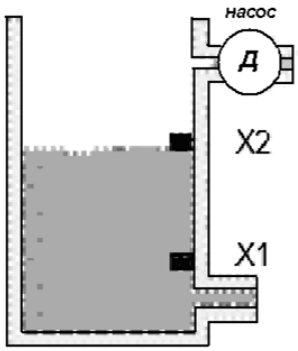



Рисунок 1.46 – Бак с водой

Когда вода опустится ниже уровня второго датчика, то оба входа (вход (X1) и вход (X2)) изменят свое состояние с «0» на «1», за этим последует изменение состояния внутреннего реле M1 с «0» на «1» и включение насоса.

Когда вода поднимется до уровня второго датчика, то вход (X2) будет в состоянии «1», а вход (X1) изменит свое состояние с «1» на «0», и в результате насос будет продолжать работать, так как внутреннее реле будет в состоянии «1» до тех пор, пока вода не достигнет уровня верхнего датчика и не изменит состояние входа (X2) с «1» на «0». За этим последует изменение состояния регистра M1 с «1» на «0» и выключение насоса.

Данный пример демонстрирует необходимость и важность использования внутренних реле контроллера, количество которых зависит

только от модели контроллера. Внутренние реле позволяют решать задачи управления с минимальным числом внешних устройств, что и является основной задачей контроллера.

## 1.17 **Программирование счетчика. Команда COUNTER**

Счетчик – самое простое устройство в контроллере, так как предназначен только для одного – считать. Конечно, в реальности в зависимости от модели контроллер может поддерживать различные варианты использования данной функции:

- суммирующие счётчики (прямой счет 1,2,3 ...);
- обратные счётчики (счет вниз 3,2,1 ...);
- счет вверх-вниз (счет вниз 1,2,3,4,5,4,3,2,3,4,5 ...).

Возможность использования того или другого типа счетчика относится только к возможностям конкретной модели контроллера.

Дополнительно счетчики делятся по способу обработки импульсов на два основных типа:

- Программные – напрямую зависят от быстродействия контроллера и не могут работать быстрее скорости обработки двух программных циклов (при использовании программного счетчика допускается, что быстродействии счетчика не превышает «*Времени цикла обработки»* X2. В противном случае необходимо использовать высокоскоростные - аппаратные счетчики);
- Аппаратные – не зависят от быстродействия контроллера и могут работать быстрее времени обработки одного программного цикла (с частотой до 100 КГц). Мы можем считать, что данный тип счетчиков физически существует.

Независимо от того – программные или аппаратные счетчики, выбор должен определяться только тем, с какой скоростью будет работать *счетный вход* и позволяет ли быстродействие контроллера обработать сигналы с датчика или нет.

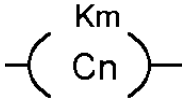
Для правильного использования счетчиков необходимо определить для себя всего 3 вещи:

- **С какой частотой мы хотели бы считать.** Так как обычно, использование *программного счетчика* допустимо для любого из входов, то при выборе *аппаратного счетчика* мы можем использовать только те входы, которые служат для высокоскоростного счета. Для определения возможности использования того или иного входа в качестве высокоскоростного необходимо сверяться с руководством пользователя или с руководством по программированию;

▪ **До какого значения мы собираемся считать импульсы.** Диапазон, в котором может считать контролер 0 ... 32.767, -32.768 ... -32.768, 0 ... 65535, ... зависит только от конкретной модели контроллера;

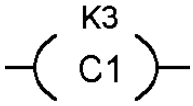
▪ **По какому условию мы можем остановить счет.**

**Команда (OUT Cn Km) – Инициализация счетчика.**




**C** – обозначение счетчика;
n – число от 1 до 256 – номер счетчика;
**K** – обозначение константы;
m – число от 1 - 2.147.483.648, до которого будет вестись счет.

Например, при n = 1, m = 3 :




Вспомним наш первый пример и попробуем решить следующую задачу. Пусть насос (выход (Y1)) включится только после того, как мы три раза включим-выключим вход (X2) (см. рисунок 1.47).

▪ То есть при включении входа (X1) – счетчик (C1) установится в значение «0»;

▪ Первое нажатие на вход (X2) – счетчик (C1) сравнит «1» = K3 («3») – если нет, то запомнит «1»;

▪ Второе нажатие на вход (X2) – счетчик (C1) прибавит «1», сравнит «2» = K3(«3») – если нет, то запомнит «2»;

▪ Третье нажатие на вход (X2) – счетчик (C1) прибавит «1» сравнит «3» = K3(«3») – если да, то сменит состояние C1 с «0» на «1» и насос заработает;

▪ Повторное включение Входа (X1) – счетчик (C1) установится в значение «0» и цикл начнется с начала.

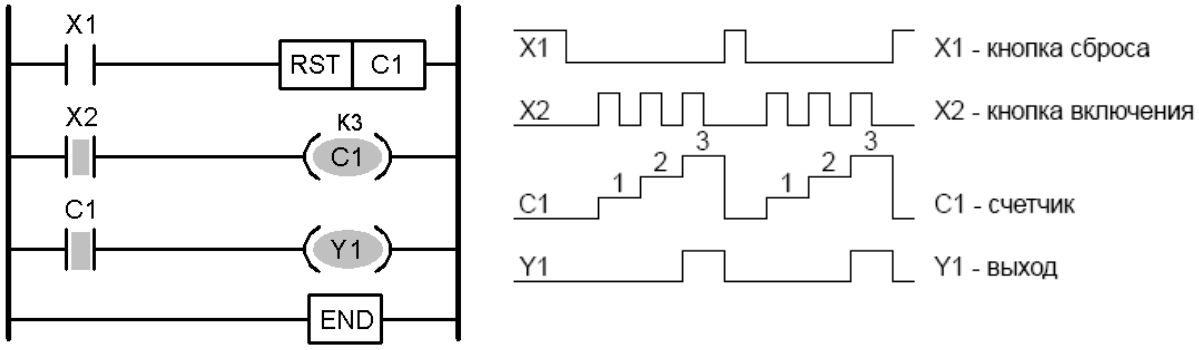



Рисунок 1.47 – Схема и диаграмма работы счетчика

## 1.18 **Программирование таймера. Команда TIMER**

Попробуем классифицировать, какими бывают таймеры:

- ***С задержкой по включению.*** Другими словами, после того, как Вы нажали на выключатель (т.е. придя поздно вечером домой и нажав на выключатель, Вы с удивлением обнаруживаете, что свет загорается только через одну минуту после включения выключателя);
- ***С задержкой по выключению.*** Иначе, после того, как Вы выключили свет (т.е. уходя поздно вечером из дома и нажав на выключатель, вы с удивлением обнаруживаете, что свет в квартире погас только спустя некоторое время);
- ***Накапливающий таймер*** – Этот тип таймера позволит Вам выключить свет только после того, как суммарное время включения достигнет определенного значения. Данный тип таймера требует обязательного использования двух входов.

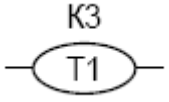
Что мы должны определить для себя – это всего лишь 2 вещи:

- Какой из входов запустит таймер;
- Какую задержку времени установить, т.е. сколько времени пройдет, прежде чем включится-выключится выход.

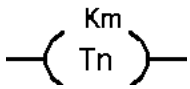
С того момента, когда все команды перед символом таймера примут значение – «истина», таймер начинает отсчёт времени и по достижении установленного значения переключит состояния выхода с «включено» на «выключено» или наоборот. В любой момент времени работы таймера есть возможность отображать его текущее состояние. Диапазон, в котором может работать контроллер (0 ... 32.767, -32.768 ... -32.768, 0 ... 65535) зависит только от конкретной модели контроллера.

Правила и возможность использования определяются документацией по программированию контроллера. Так же, как и в случае со счетчиками, некоторые модели контроллеров позволяют использовать *высокоскоростные таймеры*.

**Команда (OUT Tn Km) – Инициализация таймера.**



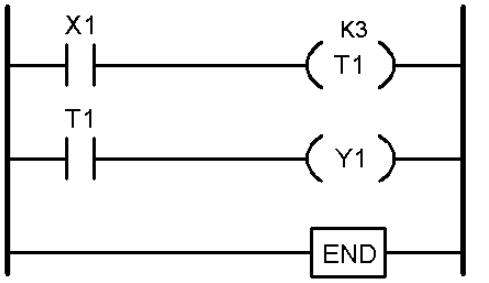

**T** – обозначение таймера;
n – число от 1 до 256-номер таймера;
**K** – обозначение константы;
m – число от 1 до 32768, до которого будет вестись отсчет времени.



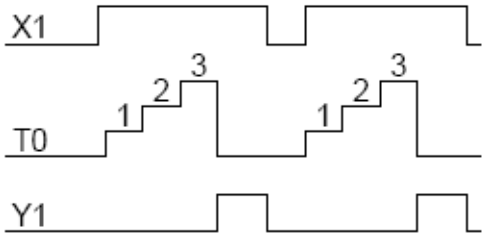

Например, при n = 1, m = 3:

**Пример 1: Таймер с задержкой по включению.**

Вспомним наш пример с уровнем воды в баке и решим следующую задачу: пусть насос (выход (Y1)) включится через три секунды после включения входа (X1). Схема работы контроллера показана на рисунке 1.48.




- Вход (X1) включился – таймер (T1) начал отсчет времени;
- Прошло три секунды – выход (Y1) включился;
- Вход (X1) выключился – выход (Y1) выключился.



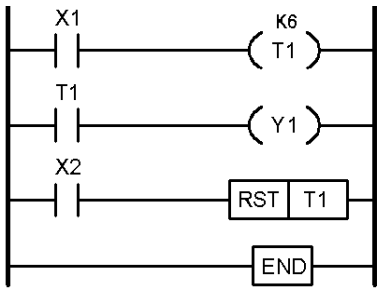

- Вход (X1) включился – таймер (T1) начал отсчет времени;
- Прошло три секунды – выход (Y1) включился;
- Вход (X1) выключился – выход (Y1) выключился.

Рисунок 1.48 – Схема и временная диаграмма программирования таймера с задержкой по включению

Понятно, что шаг мог бы быть и 0,1 мсек или 0,01 мсек и т.д.

Аналогично работает таймер с задержкой по выключению.

**Пример 2: Таймер - с накоплением**



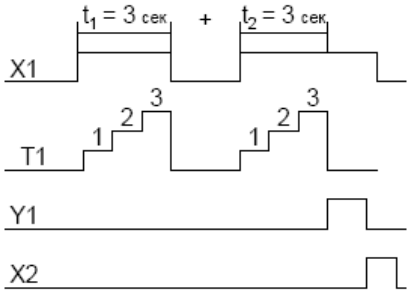




Рисунок 1.49 – Схема и временная диаграмма программирования таймера

Пусть насос (выход (Y1)) включится только после того, как вход (X1) отработает цикл включено-выклю-чено-включено, общей длительностью 6 секунд. Схема работы контроллера показана на рисунке 1.49.

- Вход (X1) включился – таймер (T1) начал отсчет времени;
- Прошло три секунды – выход (X1) выключился;
- Таймер запомнил время работы входа(X1) – t1 = 3 сек;
- Вход (X1) включился – таймер (T1) продолжил отсчет времени, как только T = t1 + t2 = 6 сек, то выход (Y1) включился;

| | |
|---|---|
| с накоплением | ▪ Вход (X2) включился – таймер (T1) изменил свое состояния с «1» на «0» и выход (Y1) выключился. |

**Пример 3: Таймер с памятью**

Наряду с уже описанными видами таймеров имеются также таймеры с памятью, которые после отключения управляющей логической связи сохраняют уже накопленное значение времени. Действительное значение времени в таймере записывается в память, содержимое которой сохраняется и при отключении напряжения. Пример работы таймеров с памятью показан на рисунке 1.50.

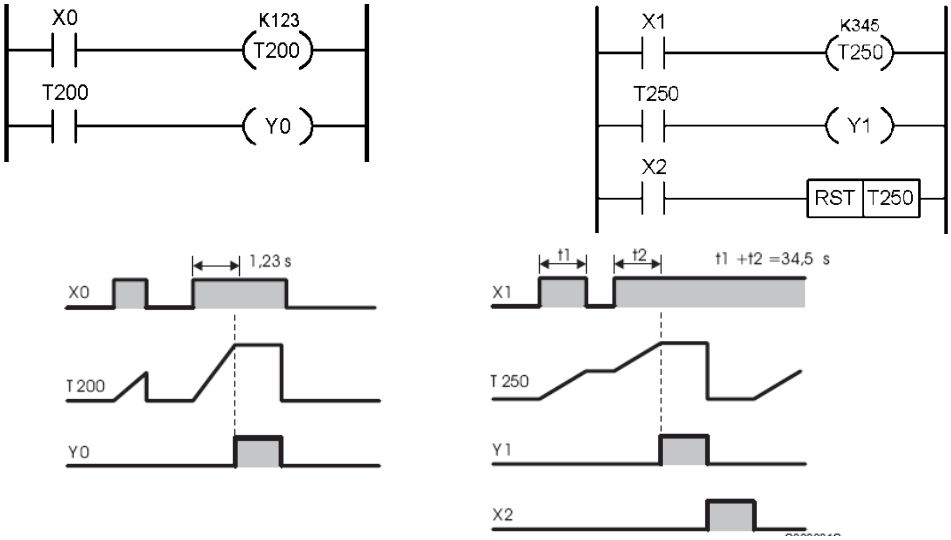



Рисунок 1.50 – Схемы и временные диаграммы работы таймера с памятью

TIMER – погрешность

Теперь вернемся немного назад и вспомним, как работает контроллер – что такое быстродействие? Конечно, если использовать таймер с шагом 1 секунда, то можно не беспокоиться, но когда используются таймеры, которые работают с шагом от 0,001 секунды, просто необходимо помнить о быстродействии контроллера.

Рассмотрим два типичных случая, в которых обычно не учитывается погрешность при использовании таймера:

- погрешность по входу;
- погрешность по выходу.

Погрешность по входу (рисунок 1.51) означает, что с момента, когда включится вход (X1), до начала работы таймера пройдет время, равное:

$$t_{Вып.прог} \cdot 2 + t_{Вх} + t_{Вых} = t_{Погр.по\ Вых.}$$

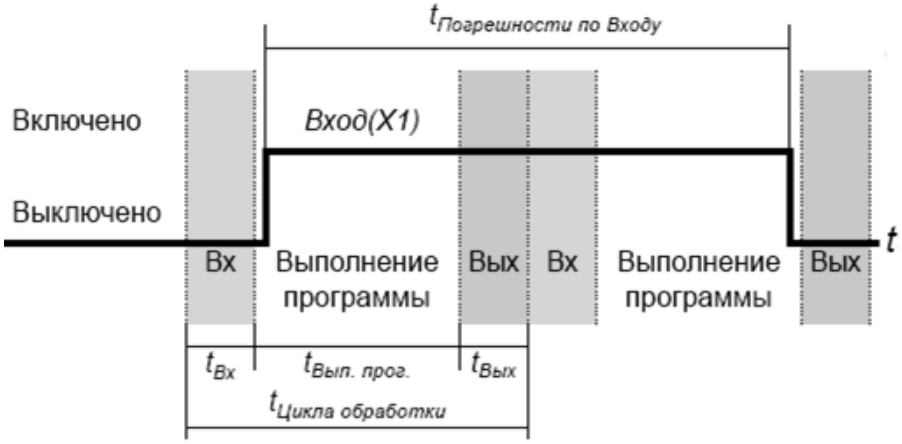



Рисунок 1.51 – Временная диаграмма погрешности таймера по входу

Погрешность по выходу (рисунок 1.52) означает, что с момента, когда выключится вход (X1), до начала работы таймера пройдет время равное:

$$t_{Вып.прог} \cdot 2 + t_{Вх} + t_{Вых} = t_{Погр.по\ Вых.}$$

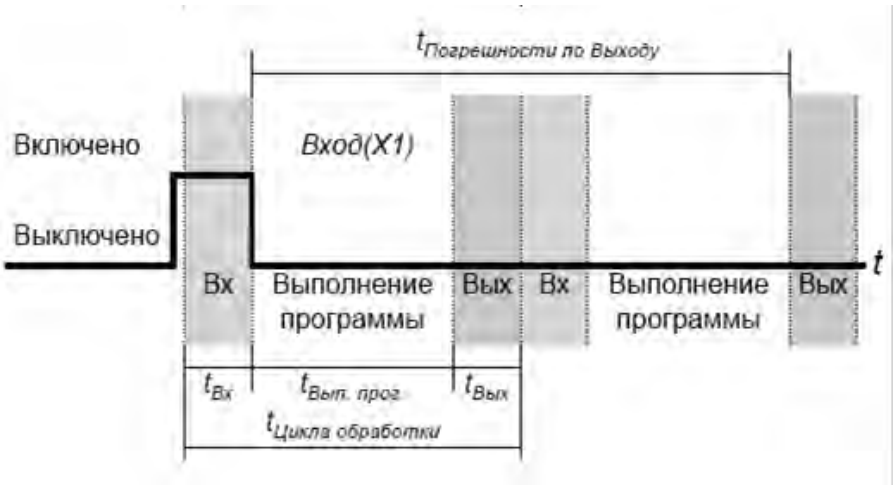
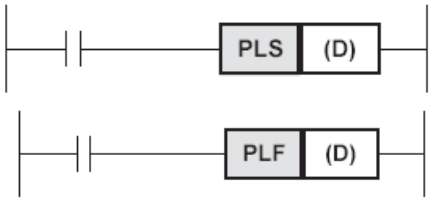



Рисунок 1.52 – Временная диаграмма погрешности таймера по выходу

Таким образом, если контроллер имеет $t_{\text{Цикла обработки}}$ = 5 мсек и поддерживает работу таймера с шагом 1 мсек., то минимальный шаг, с которым может корректно работать таймер контроллера, должен быть больше, чем 10 мсек. В действительности, кроме «программной погрешности», мы должны принимать во внимание и существование «аппаратной погрешности», т.е. времени, необходимого контроллеру на проверку действительности срабатывания входа. В реальных условиях возможен шум или скачок, который контроллер может принять за включение входа, хотя этого и не произошло. Обычно производители предусматривают настройку данного параметра в диапазоне от 0 до 10 мсек., в зависимости от «чистоты» линии.

## 1.19 **Программирование одиночных импульсов. Команды (PLF) и (PLS)**




Генерация одного импульса – опознание фронта сигнала независимо от продолжительности входного сигнала для включения соответствующего операнда (выхода Y или внутреннего реле M). Операнд остается включенным на протяжении одного цикла программы (скана).

PLS – генерация одиночного импульса по возрастающему фронту входного сигнала.

PLF – генерация одиночного импульса по спадающему фронту входного сигнала.

Пример применения однократных импульсов показан на рисунке 1.53:

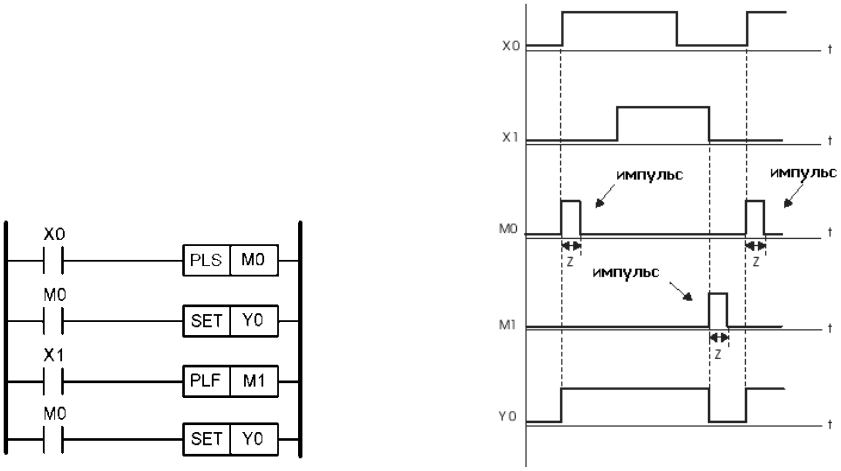



Рисунок 1.53 – Схема и временная диаграмма применения команды временных импульсов

При возрастании входного сигнала на входе X0 с «0» до «1» (возрастающий фронт) внутреннее реле M0 благодаря «PLS»-команде получает импульс (включается на время одного цикла программы). С помощью этого импульса по контакту реле M0 включается выход Y0. Лишь когда на входе X1 пройдет смена сигнала с «1» на «0» (падающий фронт), выход Y0 снова отключится (рисунок 1.53).

Генерация одиночного импульса по возрастающему фронту входного сигнала

Генерация одиночного импульса по спадающему фронту входного сигнала

Z – Время цикла программы (время скана)

## 1.20 **Инструкции процесса отработки программы**

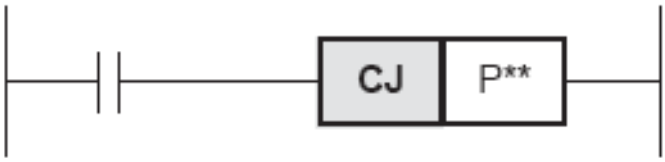
### 1.20.1 Структуризация программы

Программы для ПЛК составляются из трех основных элементов: главная программа, подпрограммы (необязательные) и программы обработки прерываний (необязательные).

- Главная программа. В этой основной части программы располагаются операции, управляющие всем приложением. Операции главной программы в каждом цикле обрабатываются последовательно. Для окончания главной программы используется инструкция абсолютного завершения программы («END»).
- Подпрограммы. Эти необязательные компоненты программы обрабатываются только тогда, когда они вызываются из главной программы. Подпрограммы располагаются после главной программы (после «FEND»-инструкции и перед «END»-инструкцией). Каждая подпрограмма завершается командой «SRET», «вернуться».
- Программы обработки прерываний. Эти необязательные компоненты программы обрабатываются только тогда, когда появляется событие прерывания. Программы обработки прерываний располагаются также после главной программы (после «FEND»-инструкции и перед «END»-инструкцией). Каждая программа обработки прерываний завершается операцией «IRET», «вернуться из программы обработки прерываний».

Возможно располагать подпрограммы и программы обработки прерываний после главной программы в смешанной последовательности. Однако, если Вы хотите, чтобы программа была легко читаемой и понятной, то необходимо присоединить все подпрограммы непосредственно к главной программе, а затем расположить все программы обработки прерываний вслед за подпрограммами.

### 1.20.2 Переход внутри программы (CJ)




С помощью «CJ»-инструкции можно «перепрыгивать» через часть программы. При применении этой инструкции время программы может уменьшиться. Цель (конец) перехода определяется установкой маркировки (маркировка точки) в программе. Для маркировки используют точки P0…P63.

Если внутри подпрограммы перехода программируется инструкция сброса (отключения) для счетчика с запоминанием, то процесс сброса

(стирание накопленного значения) имеет место тогда, когда перепрыгивается цепь схемы катушки счетчика.

Примечание:

При дублировании записи выходов необходимо следить за тем, чтобы оба выхода никогда не были активными в одно и тоже время. Это может привести к ошибочной отработке программы.

Маркировка точки в программе:

При программировании на языке контактных схем маркировка точки указывается слева от цепи схемы (рисунок 1.54).

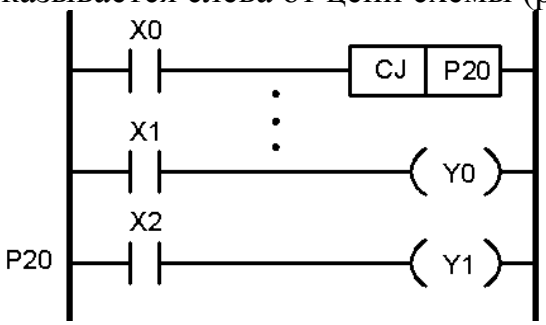



Рисунок.1.54 – Пример программирования CJ-инструкции

Если включается X0, то выполняется переход к маркировке точки P20.

Пример двукратной вставки в программу адреса точки P9 показан на рисунке 1.55.

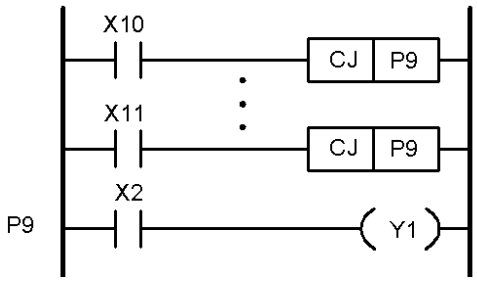



Рисунок 1.55– Пример двукратной вставки в программу адреса точки P9.

Если X10 включен, то выполняется переход к промаркированной точке P9.
Если X10 выключен, а X11 включен, то все равно произойдет переход к точке P9.

Примечание:

Одинаковая маркировка точек не должна многократно использоваться в программе, т.к. может создаться ошибка в работе программы.

Маркировка точки перед «CJ»-инструкцией перехода:

Обратный переход (вверх программы) также может выполняться внутри программы.

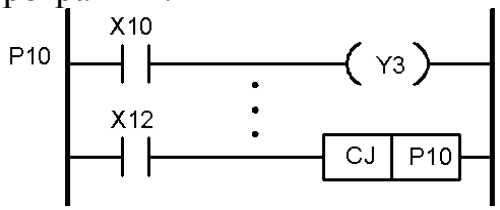
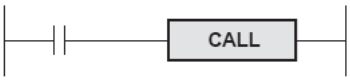



Рисунок.1.56 – Маркировка точки перед «CJ»-инструкцией

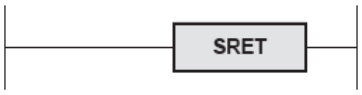
Примечание:

Если входной сигнал для «CJ»-инструкции держится больше 200 мс, то появляется ошибка времени работы (Watch-Dog-Timer).

### 1.20.3 Вызов подпрограммы (CALL / SRET)




Вызов подпрограммы




Конец подпрограммы

С помощью «CALL»-инструкции вызывается подпрограмма, промаркированная с помощью точек (P0...P62). Подпрограмма программируется после «FEND»-инструкции и перед «END»-инструкцией. В конце подпрограммы должна находиться «SRET»-инструкция. Если активируется «CALL»-инструкция, то выполняется переход к указанной точке маркировки. После отработки «SRET»-инструкции выполняется обратный переход к инструкции, следующей за «CALL»-инструкцией. Активированные в подпрограмме операнды остаются таковыми после отработки подпрограммы до новой ее обработки. В подпрограмме должны использоваться таймеры T192...T199 и T246...T249. Те же точки могут использоваться с любым числом CALL-инструкций. Внутри подпрограммы могут вызываться другие подпрограммы. Возможно максимум 4 уровня ветвления.

Пример применения подпрограмм показан на рисунке 1.57.

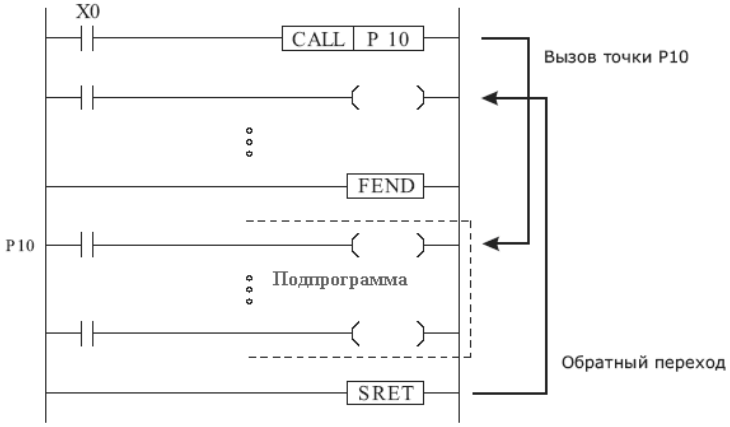
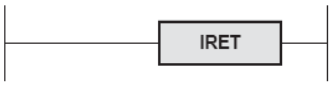
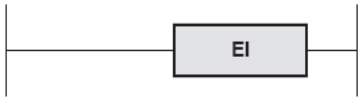



Рисунок 1.57 – Пример программирования с применением «CALL»- и «SRET»-инструкций

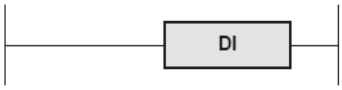
### 1.20.4 Ввод прерывания программы (IRET, EI, DI)




Окончание программы прерывания




Импульсный запуск/активизация программы прерывания




Деактивация программы прерывания

Принцип функционирования.

Вызов, Окончание, Активизация и Деактивация программы прерывания.

Вызов программы прерывания.

При вызове программы прерывания останавливается основная программа и выполняется переход к программе прерывания. После окончания программы прерывания выполняется возврат к основной программе. Начало программы прерывания определяется установкой маркировки (точки прерывания). Конец программы прерывания определяется «IRET»-инструкцией. Входы X0...X5 образуют входы прерывания. Сигналы прерывания должны иметь ширину импульса минимум в 200 мкс.

Примечание:

Входы X0...X5 не могут применяться одновременно для обработки сигналов прерывания и для обработки сигналов высокоскоростного счетчика.

Адресация точек прерывания.

Точка прерывания: I X 0 x

X – адрес 0...5, соответствующий входам X0...X5

x = 0 – прерывание при падающем фронте входного сигнала.

x = 1 – прерывание при возрастающем фронте входного сигнала.

Примечание:

Адрес прерывания может использоваться только один раз

Применение «EI»- и «DI»-инструкций.

С помощью «EI»-инструкции могут активироваться инструкции прерывания. Это означает, что после отработки «EI»-инструкции, смена сигнала, которая появляется на одном из входов X0...X5, обрабатывается как сигнал прерывания в программе.

С помощью «DI»-инструкции могут деактивироваться инструкции прерывания. Это означает, что после отработки «DI»-инструкции, смена сигнала, которая появляется на одном из входов X0...X5, не обрабатывается больше как сигнал прерывания в программе.

Примечание:

Если ни одна из обеих инструкций «EI» или «DI» не программируется, режим прерывания не активизируется, т.е. тогда не может обрабатываться никакой сигнал прерывания.

Отработка программы прерывания.

Во время исполнения программы прерывания не может вызываться никакая другая программа прерывания. Однако может программироваться два уровня разветвления.

Несколько, одна за другой следующие, программы прерывания обрабатываются в последовательности их вызова. Если одновременно вызываются несколько программ прерывания, то вначале обрабатывается программа прерывания с более низким адресом точки.

Выключение прерывания.

Любое прерывание может повременно или постоянно выключаться посредством включения соответствующего специального меркера (внутреннего реле). Для FX0S первым специальным меркером является M8050, который выключает прерывание IOab. Соответствующие специальные меркеры указываются в руководстве пользователя.

Пример адресации точки прерывания показан на рисунке 1.58.

Точка I001 – вход прерывания X0, прерывание при возрастающем фронте входного сигнала (смена сигнала с «0» на «1»).

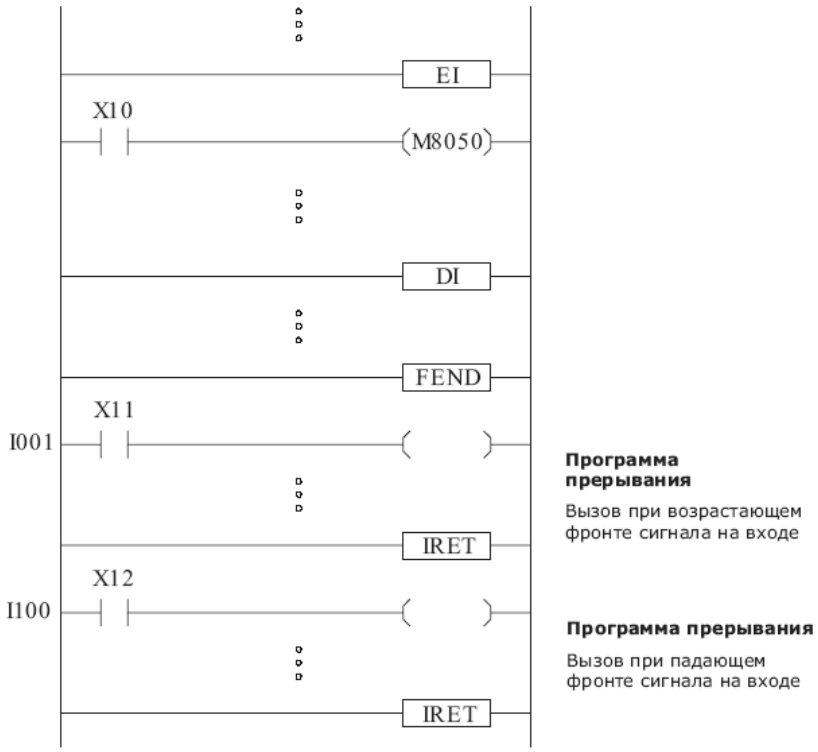



Рисунок 1.58 – Пример программирования при использовании инструкций «EI», «DI» и «IRET»

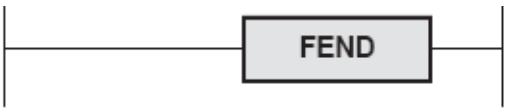
Если вход X0 устанавливает сигнал прерывания во время выполнения шага программы внутри области от EI-инструкции до DI-инструкции, то имеет место переход к программе прерывания I001. Программа прерывания выполняется и происходит возврат в основную программу.

Программа прерывания I001 не выполняется, если активизирован специальный меркер M8050 (вход X10 включен).

Если вход X1 устанавливает сигнал прерывания во время выполнения шага программы внутри области от EI-инструкции до «DI»-инструкции, то имеет место переход к программе прерывания I100. Программа прерывания выполняется и происходит возврат в основную программу.

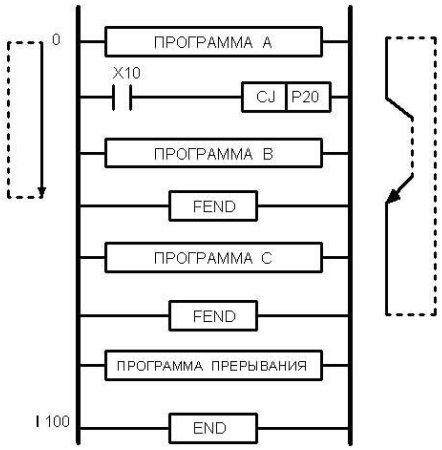
Если появляются одновременно сигналы X0 и X1, то вначале обрабатывается программа прерывания I001, а затем программа прерывания I100.

### 1.20.5 Конец области программы (FEND)




С помощью «FEND»-инструкции определяется конец области программы. В программе возможно применение нескольких «FEND»-инструкций. После отработки «FEND»-инструкции выполняется обработка выходов, затем выполняется возврат к программному шагу 0, после чего обновляется обработка входов и время установки контроля цикла программы. Пример работы программы с применением «FEND» показан на рисунке 1.59.

Работа программы, если X10 не включен

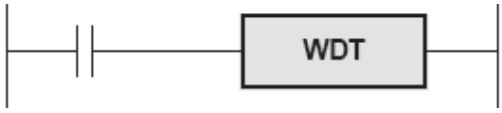



Работа программы, если X10 включен.

Область программы B перепрыгивается

Рисунок.1.59 – Пример программирования «FEND»-инструкции

### 1.20.6 Обновление таймера времени работы программы (WDT)




С помощью «WDT»-инструкции можно длинные программы разделить на отдельные отрезки программ. Время цикла программы (скана) определяется для каждого отдельного отрезка программы самим ПЛК (WDT обновляется после каждого отрезка программы). С помощью «WDT»-инструкции можно обрабатывать программу, время цикла которой превышает

200 мс. На рисунке 1.60 показана типичная ситуация, в которой необходимо использовать «WDT»-инструкцию.

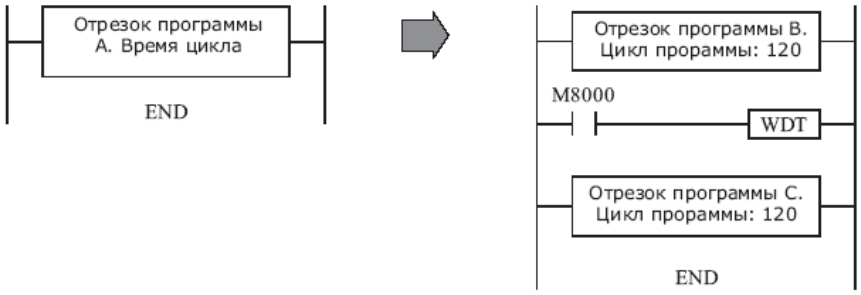



Рисунок 1.60 – Пример программирования при использовании «WDT»-инструкции

Время обработки для отрезка программы A превысило значение 200 мс. Поэтому отрезок программы A был разделен с помощью «WDT»-инструкции на два отрезка программ (B, C). Отрезки программ B и C требуют соответственно только по 120 мс времени цикла.

Значение времени цикла программы изменяется в специальном регистре D8000. Если время цикла программы постоянно превышает значение 200 мс, можно изменить значение максимально допустимого времени цикла в специальном регистре D8000 (смотри пример на рисунке 1.61)

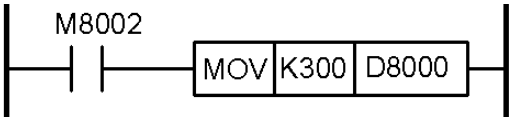
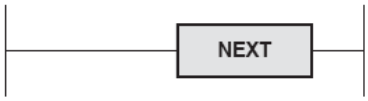



Рисунок 1.61 – Пример изменения длительности скана программы

Установка максимально допустимого времени цикла программы в регистре данных D8000 на значение 300 мс.

### 1.20.7 Повторение части программы, задание цикла (FOR, NEXT)


Начало повторения программы




Конец повторения программы

Программирование повторений программы (петля программы). Эти инструкции позволяют, чтобы часть программы между «FOR»- и «NEXT» повторялась «n» раз. После завершения «FOR» выполняется шаг программы после «NEXT»-инструкции. Значение «n» может находиться внутри следующей области значений: от +1 до +32 767. Если для «n» указано значение между 0 и -32 767, то петля «FOR-NEXT» отрабатывается только один раз. Можно программировать до пяти «FOR-NEXT»-уровней вложения.

Примечание:

«FOR»- и «NEXT»-инструкции могут программироваться только попарно. К каждой инструкции «FOR» должна программироваться соответственно «NEXT»-инструкция.

Источники ошибок:

В следующих случаях появляются ошибки в работе программы:

• «NEXT»-инструкция запрограммирована перед «FOR»-инструкции.

• «NEXT»-инструкция запрограммирована после «FEND»-инструкции или «END»-инструкции.

• Количество «NEXT»-инструкций не соответствует количеству «FOR»-инструкций.

Пример применения «FOR»- и «NEXT»-инструкций показан на рисунке 1.62.

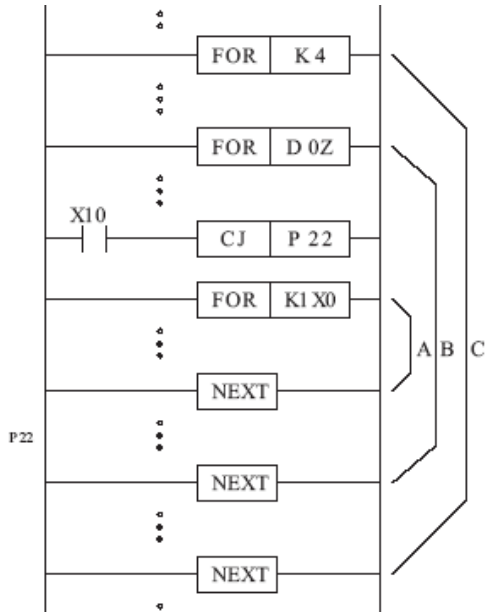



Рисунок 1.62 – Использование «FOR»-
и «NEXT»-инструкций

В примере запрограммированы три входящие друг в друга «FOR»- и «NEXT»-уровня вложения.

• Отрезок программы C обрабатывается четыре раза (здесь K4 константа). В конце обработки последний программный шаг выполняется после третьей «NEXT»-инструкции.

• При каждом исполнении отрезка C отрезок программы B обрабатывается шесть раз, если в регистре данных D0Z записано число 6.

• Поэтому отрезок B обрабатывается 6 x 4 = 24 раза.

• Если вход X10 включен, то «FOR-NEXT»-петля (отрезок программы) пропускается (не обрабатывается) с помощью «CJ»-инструкции.

• Если вход X10 выключен и содержание K1X0 (блок K1 - первых 4 бита – в слове X0) равно 7, то при каждом выполнении отрезка B отрезок программы обрабатывается семь раз.

• Поэтому отрезок A обрабатывается 168 раз. (6 x 4 x 7)

# 2.ПРОГРАММИРОВАНИЕ УСТРОЙСТВ ЧИСЛОВОГО ПРОГРАММНОГО УПРАВЛЕНИЯ

## 2.1 Базовые понятия

**Кадры программы.** Система ЧПУ исполняет кадры программы последовательно, один за другим. Каждый кадр состоит из некоторой совокупности слов, которые, в свою очередь, содержат адресную часть и цепочку цифр. К примеру, кадр может состоять из девяти слов с адресами N_G_ X_Y_Z_F_S_T_M. Последовательность полноформатных слов выглядит, например, так: GOO X-23450 Y40 M03 S250.

Незначащие нули цифровой части слова пропускают. Числа типа real записывают с десятичной точкой; причем, незначащие нули в дробной части также опускают.

Например, X100.500 соответствует X100.5. Число слов в кадре переменно. Слова, описывающие перемещения, могут иметь знак (+/-). При отсутствии знака перемещение полагается положительным.

**Модальный эффект**. Большинство слов модальны. Это означает, что они остаются в силе на протяжении нескольких кадров, пока значение слова не изменится, или пока функция, представленная словом, не будет выключена. Пусть, например, с помощью функции G1 запрограммирована линейная интерполяция с некоторой скоростью подачи. В последующих кадрах эта функция сохранит свою активность, пока интерполяция не изменится на круговую (функция G2) или линейную с ускоренной подачей (функция GO). Слова, которые действуют только в своем кадре, - немодальны.

Слова имеют смысл **инструкций** (например, при задании типа перемещений вдоль координатных осей X, Y, Z, C) или **специальный функций** (например, при назначении подачи, частоты вращения и др.).

**G-адреса.** G-адреса используют, например, для программирования типа перемещения (с линейной или круговой интерполяцией, и др.). Слова с G-адресами относятся к числу инструкций, которые называют подготовительными функциями. Подготовительные функции разбиты на группы; причем функции из разных групп взаимно независимы. С другой стороны, G-функции одной и той же группы взаимно модальны, т.е. действуют до отмены или замены G-функцией из той же группы. В кадре может быть представлена только одна G-функция из своей группы.

**Адреса X, Y, Z, C и др.** Эти адреса используют для обозначения координатных осей, вдоль которых осуществляются перемещения. Пример: N G60 X10 Y10 B135; где X, Y – координатные оси подачи; B1 – ось вспомогательных перемещений.

**Специальные функции**. Примерами адресов специальных функций

могут послужить: F (подача), S (частота вращения шпинделя), M (вспомогательная функция; связанная, например, с управлением электроавтоматикой), T (выбор инструмента). В примере показан кадр, в котором присутствуют позиционная информация и специальные функции: G01 X40 Y50 F250 S500 T05 M03. Здесь задано перемещение (траекторная информация); а также и специальные функции: подачи F250, частоты вращения шпинделя по часовой стрелке S500; функции инструмента T05б обеспечивающей его доступность в инструментальном магазине.

**Номера кадров.** Именем кадра, открывающим кадр слева в строке, служит его номер. Имя состоит из адреса N и собственно номера (например, N10). Нумерация облегчает чтение программы. Принято нумеровать кадры последовательно, по возрастающей степени, с приращением 10 (например, N10 N20 N30 и т.д.). При этом возникает возможность включать дополнительные кадры при редактировании программы. При ветвлениях и переходах программы номера кадров служат метками.

Номера кадров используют также и в циклах, и в подпрограммах.

**Комментарии**. Комментарии служат для пояснений и документирования. Хорошо комментированная программа служит прообразом для других программистов при любых изменениях программы. Однако каждый символ комментария увеличивает длину файла управляющей программы на один байт. Комментарии указывают в скобках или предваряют кавычками. Комментарии в скобках игнорируются системой ЧПУ, а предваряемые кавычками – визуализируются на экране монитора.

**Работа управляющей программы.** При отсутствии инструкций, управляющих потоком кадров, кадры отрабатываются последовательно один за другим. Эта последовательность может быть нарушена инструкциями: пропуска кадров, вызова подпрограмм, перехода к другим кадрам.

Если кадры программы помечены соответствующим образом (/), то система управления проигнорирует их, если активен сигнал Skip.

**Подпрограммы.** Если какая-то часть технологического процесса повторяется, ее целесообразно оформить в виде подпрограммы, которая вызывается по мере надобности. Существуют два способа вызова подпрограммы: с P адресом или без него. Синтаксис вызова подпрограммы с P-адресом выглядит так: P<имя_подпрогр>DIN; где DIN означает, что все кадры подпрограммы написаны в коде DIN66025 (ISO6983), т.е. в коде ISO-7bit.

Все перемещения, заданные в том же кадре, будут выполнены до вызова подпрограммы. Подпрограмма может иметь свои подпрограммы путем вложения (см. рис 2.1).

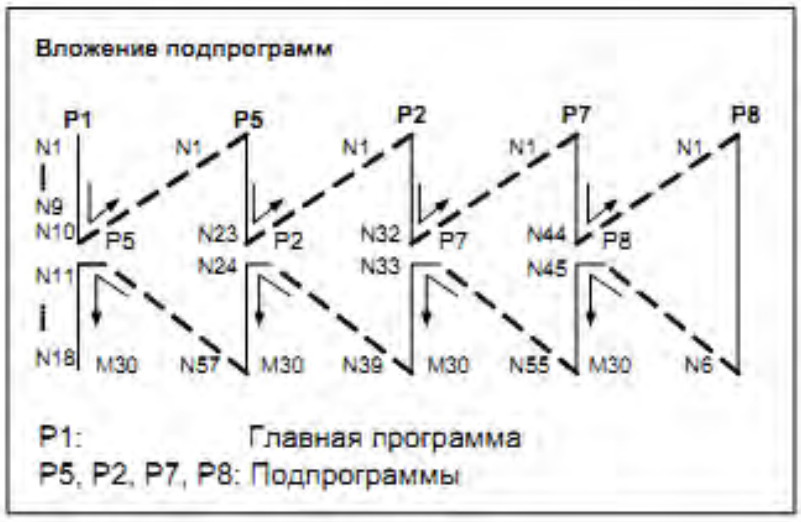



Рис 2.1. Вложение подпрограмм

Подпрограммы могут быть также вызваны под G и M адресами (об этом далее). Подпрограммы можно вызывать и без P-адреса: в этом случае достаточно указать имя подпрограммы. Кроме того, 16 G-функций зарезервированы для вызова подпрограмм.

Как правило, основная программа, кадры подпрограммы и циклы исполняются в том порядке, в каком они запрограммированы. Порядок может быть нарушен переходами, условными и безусловными. Инструкции перехода зависят от конкретной системы ЧПУ и выходят за рамки стандарта DIN 66025 (ISO 6983).

## 2.2 **Координатные оси и координатные системы**

**Физические и логические оси**. Приводы станка относятся к приводам подачи и главного движения. Приводы подачи определяют положение в рабочем пространстве станка. Различают физические и логические координатные оси. Физические оси называют также системными. Они группируются по каналам ЧПУ; причем в рамках канала координатные оси находятся в единообразном технологическом отношении друг к другу. Таким образом, группы осей могут работать (выполнять технологические операции) независимо и параллельно. Физические оси, не привязанные к каналу, называют асинхронными, или вспомогательными. Вспомогательные оси служат, к примеру, для организации перемещений в механизмах смены инструмента.

Отдельные оси внутри группы канала ЧПУ называют логическими. Они объединены интерполяционными алгоритмами, и в этой связи их называют также синхронными осями. Логические оси канала имеют индексы. Связывание физических и логических осей осуществляют при помощи так называемых «машинных параметров» станка.

**Координатная система.** Используют вправо-ориентированную координатную систему, в которой предусмотрены линейные перемещения вдоль координат X, Y и Z; каждая из которых может быть связана с круговыми вращениями поворотных осей A, B и C.

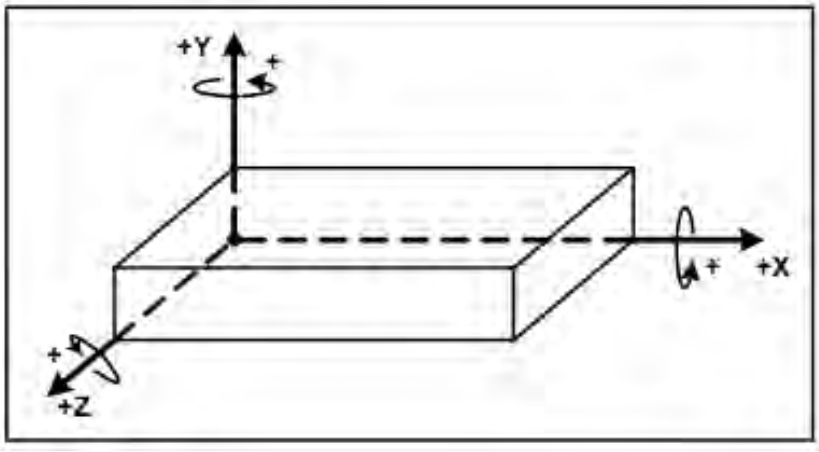



Рис.2. 2. Система координат и рабочее пространство станка

Если станок имеет единственный шпиндель, то Z-ось параллельна оси шпинделя; в противном случае она перпендикулярна плоскости зажима детали. Положительные направления осей соответствуют относительному движению инструмента и заготовки. X-ось расположена в горизонтальной плоскости соответственно плоскости зажима заготовки. Соответственно определяется Y-ось. Оси X, Y и Z являются главными. Кроме того, возможны параллельные управляемые оси, которым придают адреса U, V, W.

Поворотные движения, привязанные к базовым координатам, имеют адреса. A, B и C.

Положительное направление поворотных осей соответствует движению против часовой стрелки, если смотреть со стороны положительного направления соответствующей прямолинейной оси.

Оси, параллельные основным X, Y, Z имеют адреса U, V и W; а если существуют дополнительные параллельные координатные системы, то они имеют адреса P, Q и R.

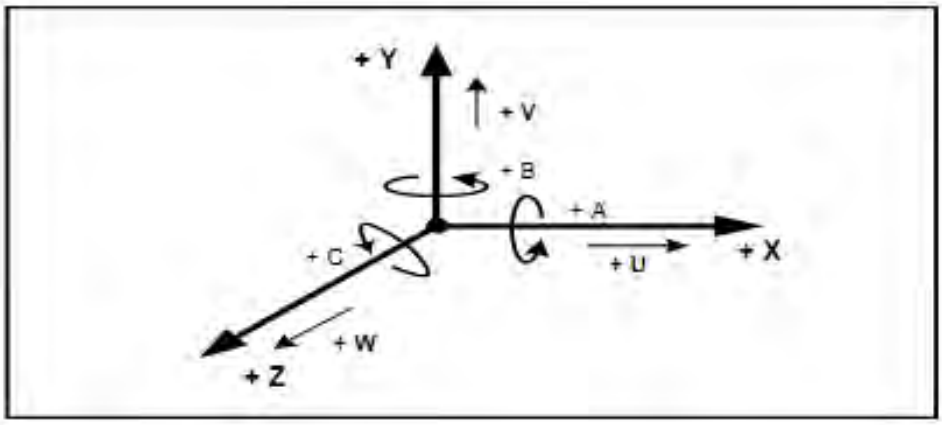



Рис.2.3.

**Координатные системы.** Для того, чтобы исполнять управляющую программу без всяких изменений по отношению к чертежу, приходится определить несколько координатных систем. Некоторые из них машинно-независимы, другие же определяются свободно. Переход от одной координатной системы к другой называется координатным переходом.

**Осевая координатная система ACS.** Совокупность осей любого канала образует «осевую координатную систему» ACS (Axes Coordinate System). Заданное движение вдоль координаты осевой координатной системы воспроизводится путем движения привода одной физической оси.

**Машинная координатная система MCS.** Осевая координатная система зависит от типа и кинематики технологической машины, а потому имеет небольшое значение при спецификации движений, связанных с обработкой деталей. По этой причине, используют так называемую MCS (Machine Coordinate System), привязанную к каналу. Как правило, эта система – Декартова; а, следовательно, не зависит от кинематики технологической машины. У каждого канала может быть своя машинная координатная система.

Ее нулевую точку **M** называют машинной и обозначают Отношение между осями машинной и осевой координатных систем называется осевой (или «обратной») трансформацией. На рис.2.4 представлены примеры

подобных отношений.

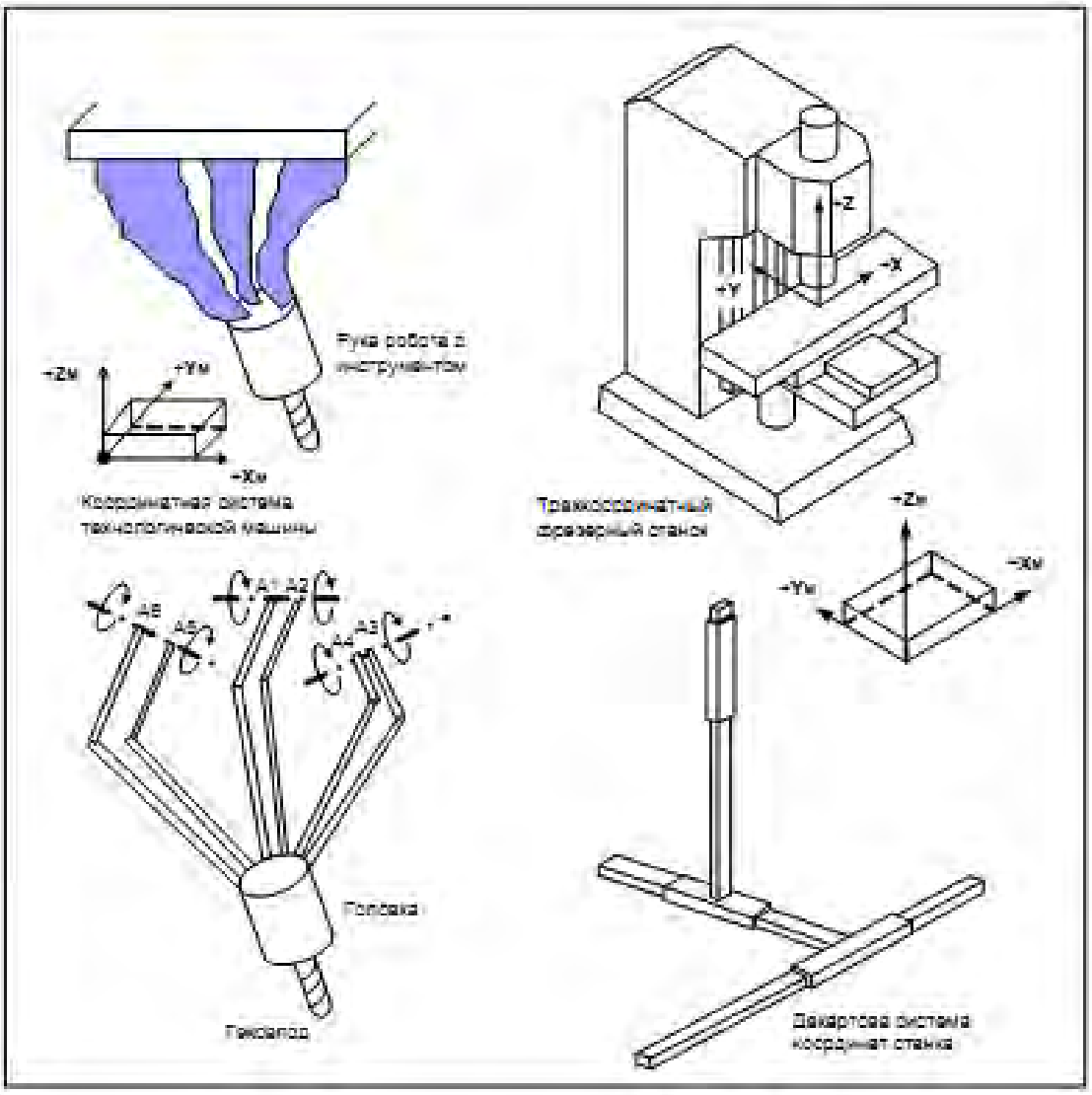



Рис.2. 4. Координатная система технологической машины и координатные системы осей

Относительную нулевую точку называют R и обозначают ⌖. Она служит для установления связи между нулем машинной координатной системы и точкой автоматического выхода в нуль следящих приводов подачи в том случае, если датчики обратной связи по положению следящих приводов работают по приращению (т.е. в относительной системе измерения). Приводы должны быть выведены в относительную точку при включении и выключении питания на станке. В этом нет необходимости, если приводы подачи располагают абсолютной измерительной системой.

**Координатная система детали WCS.** Координатную систему детали WCS (Workpiece Coordinate System) назначают свободно в зоне машинной координатной системы. Нулевую точку координатной системы детали

называют W и обозначают символом . Возможно определить несколько аддитивно связанных между собой координатных систем деталей.

**Координатная система управляющей программы PCS.** Координатной системой управляющей программы PCS (Program Coordinate System) называют такую координатную систему детали WCS, индекс которой имеет максимальное значение: $W_i$ , где i=max. Нулевую точку координатной системы PCS называют P и обозначают символом . Все запрограммированные координаты управляющей программы соотносятся с нулевой точкой P. Координатную систему PCS, как и WCS, можно свободно назначать и поворачивать в зоне машинной координатной системы WCS.

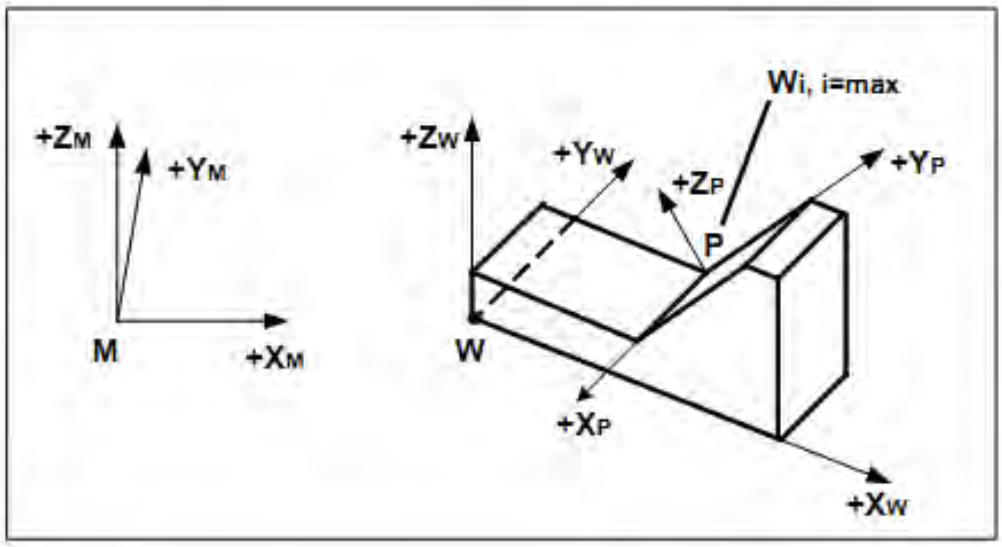



Рис.2. 5.

**Координатная система инструмента TCS.** Координатная система инструмента TCS (Tool Coordinate System) определяет положение и ориентацию инструмента в машинной координатной системе. Нулевую точку координатной системы называют **T**. Размеры инструмента (для трехкоординатного станка) задают по отношению к фиксированной точке, определяющей зажим инструмента.

В разных случаях, показанных на рис. 2.6, точка T может совпадать с точками N или E.


Рис.2. 6.

**Трансформация координат: машинные координаты, координаты детали и координаты управляющей программы**. Абсолютные значения координат обычно определены в машинной системе координат по отношению к нулевой точке M. Из практических соображений, все размеры и перемещения, указанные в управляющей программе, заданы по отношению к нулевым точкам P или W. При этом управляющие программы развязаны с машинными координатами. Благодаря программным смещениям, можно выполнять управляющую программу в любой зоне машинной системы координат без изменения размеров, указанных в управляющей программе. Если программные смещения отсутствуют, то все координаты управляющей программы интерпретируются как машинные.

Для программного смещения нуля детали предусмотрены следующие инструкции.

- G53, G54… G59. Смещение нуля ZS (Zero Shift).
- G153, G154…G159. Первое аддитивное смещение нуля ZS.
- G253, G254…G259. Второе аддитивное смещение нуля ZS.
- G160, G260, G360, G167. Смещение нуля по внешней команде.

Положение детали может быть скорректировано путем смещения нуля ее координатной системы в плоскостях (X/Y, X/Z, Y/Z) и путем поворота в плоскости (X/Y) с помощью следующих инструкций.

- G138, G139. Коррекция (компенсация) положения детали.

Для коррекции положения детали путем смещения нуля ее координатной системы и поворотов в плоскостях ( X/Y, X/Z, Y/Z) используют такие инструкции.

- G353, G354, G359. Наклон плоскости.
- G453, G454, G459. Первый аддитивный наклон.
- G553, G554, G559. Второй аддитивный наклон.

Как уже отмечалось, последняя координатная система, из серии

координатных систем детали, называется координатной системой управляющей программы. При смещении ее нуля по отношению к координатной системе детали используют следующие инструкции.

- G169, G168. Смещение нуля координатной системы управляющей программы. G269, G268. Аддитивное смещение нуля.

Иллюстрация к применению отдельных инструкций представлена на рис.2. 7.

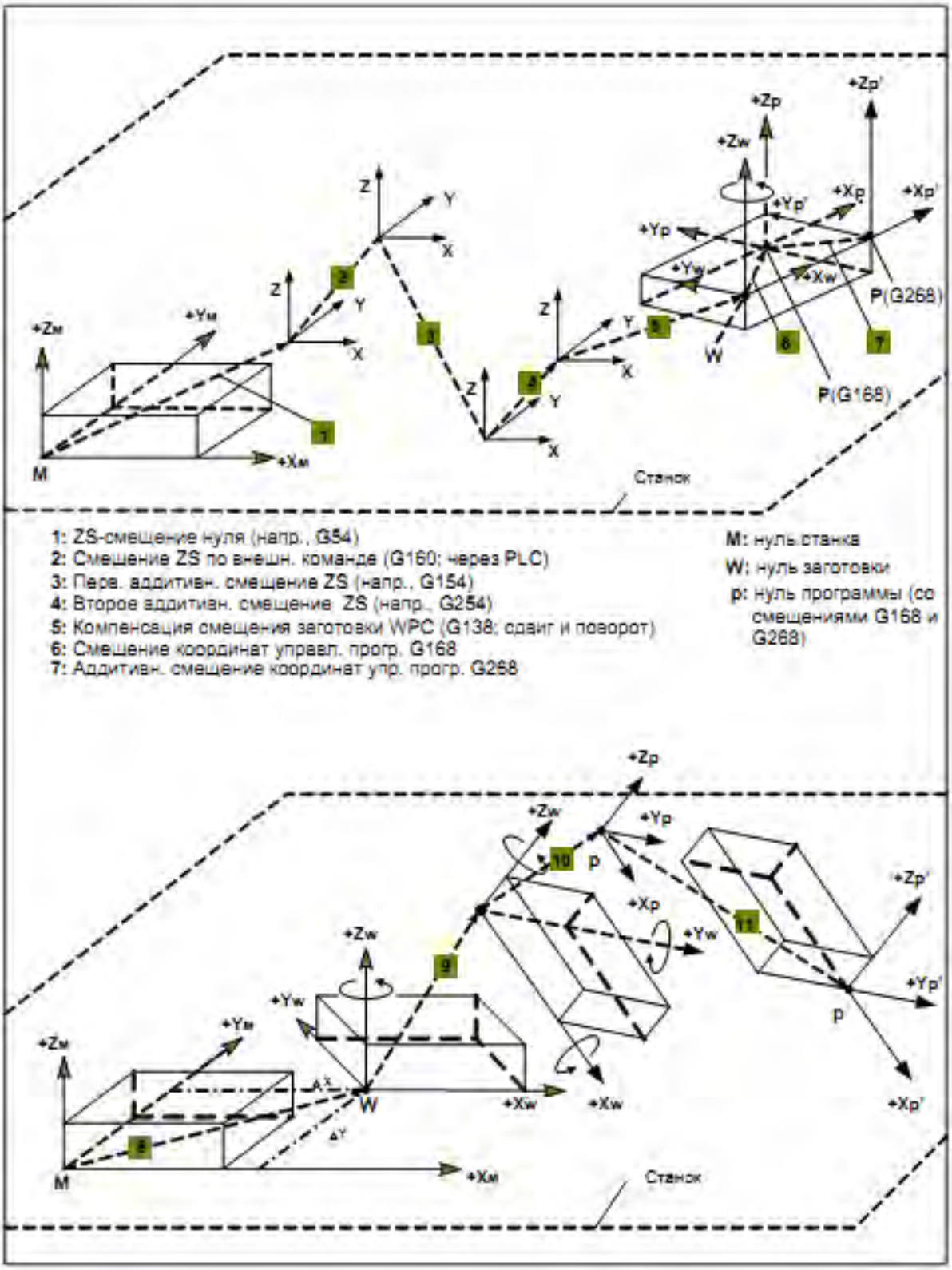



Рис. 2.7.

Активизация смещений зависит от тех или иных G-функций; она возможна при помощи «таблиц смещения нуля», при помощи первого и второго аддитивных смещений нуля ZS. Таблицы смещения нуля используют для хранения смещений между нулевой точкой M, с одной стороны, и

нулевыми точками P или W. Если соответствующее значения смещения активизировано, то это значение автоматически добавляется системой ЧПУ к каждому абсолютному значению координаты в управляющей программе. Таблицы смещения нуля представлены в файловой системе системы ЧПУ в форме ASCII файлов. Функция G22 активизирует эти таблицы в каждом канале.

Работа всех остальных G-функций рассмотрена в разделе программирования G-функций. Смещение нуля по внешней команде инициируется программируемым контроллером.

Процедура определения и сохранения смещений продемонстрирована на рис.2.8. Сохранение осуществляется путем записи смещений в таблицу.

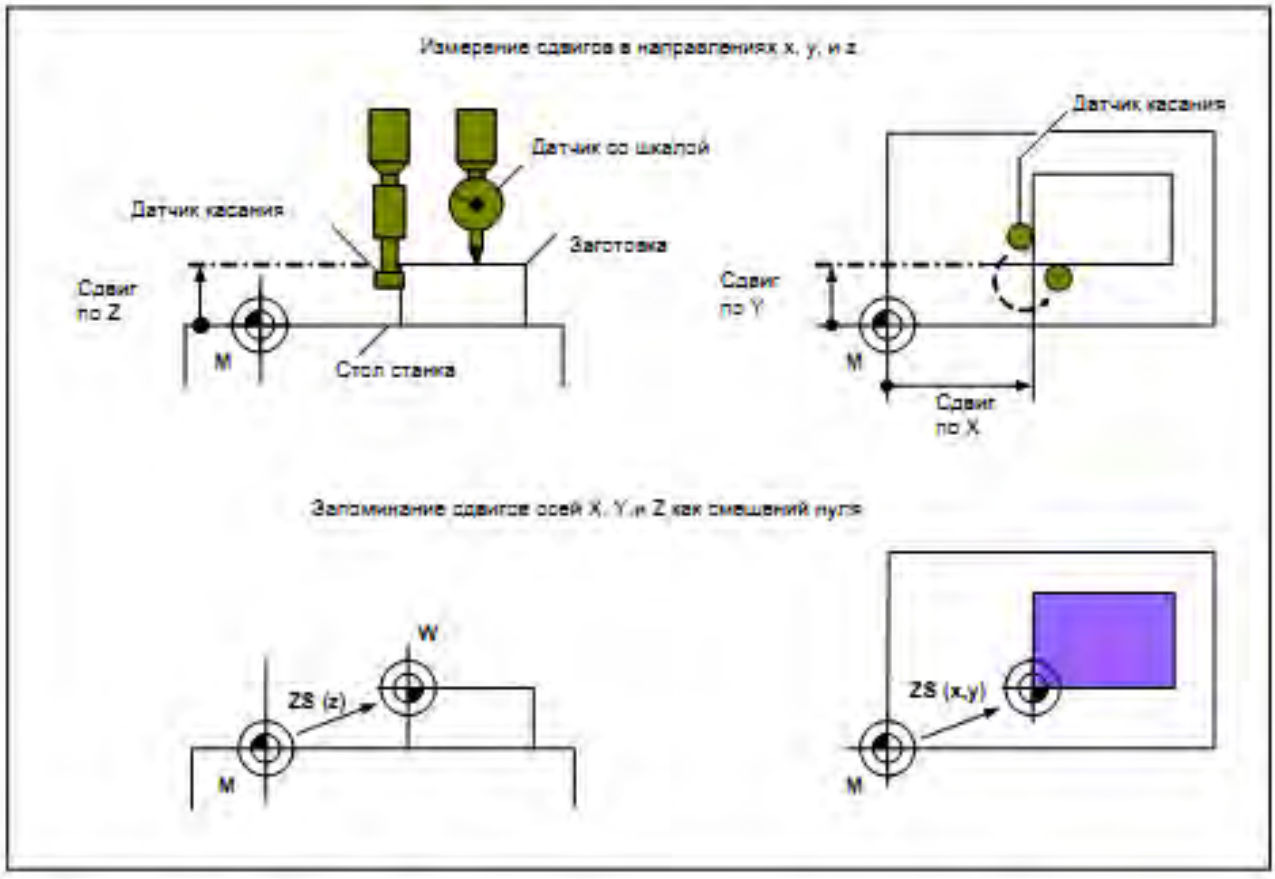



Рис.2.8.

**Функции манипулирования запрограммированным контуром**.
Возможны следующие функции манипулирования контуром:

- смещение (G60 – программирование смещения);
- зеркальное отображение, масштабирование; поворот вокруг оси, параллельной координатной оси (функции G37, G38).

Функции проиллюстрированы на рис.2.9.

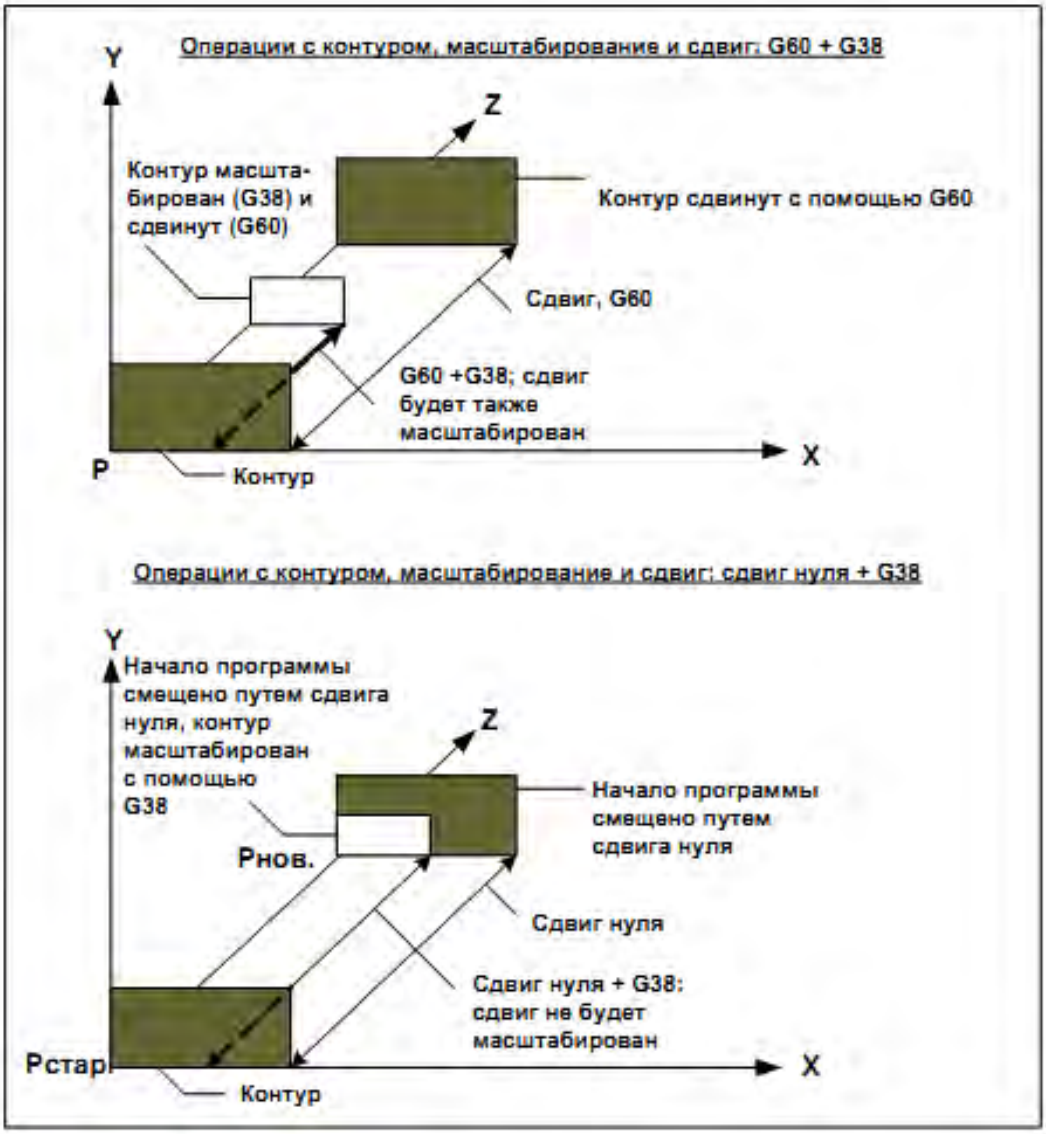



Рис. 2.9.

**Функции компенсации инструмента.** Функцию инструмента обозначают адресом **T** некоторым числом (например, слово T9 представляет собой инструмент номера 9). Инструментальный комплект состоит из инструмента и инструментальной державки.

В процессе обработки режущая кромка инструмента должна точно следовать вдоль запрограммированной траектории. В силу различия используемых инструментов, их размеры должны быть учтены и введены в систему управления перед началом воспроизведения программы. Только в этом случае траектория может быть рассчитана безотносительно к параметрам используемых инструментов. После того, как инструмент установлен в шпиндель и активизирована соответствующая коррекция (компенсация его размеров), система ЧПУ автоматически принимает в расчет эту коррекцию.

**Функции D и H компенсации инструмента.** Функция H осуществляет компенсацию длины, а функция D – компенсацию радиуса (см. рис.2.10).

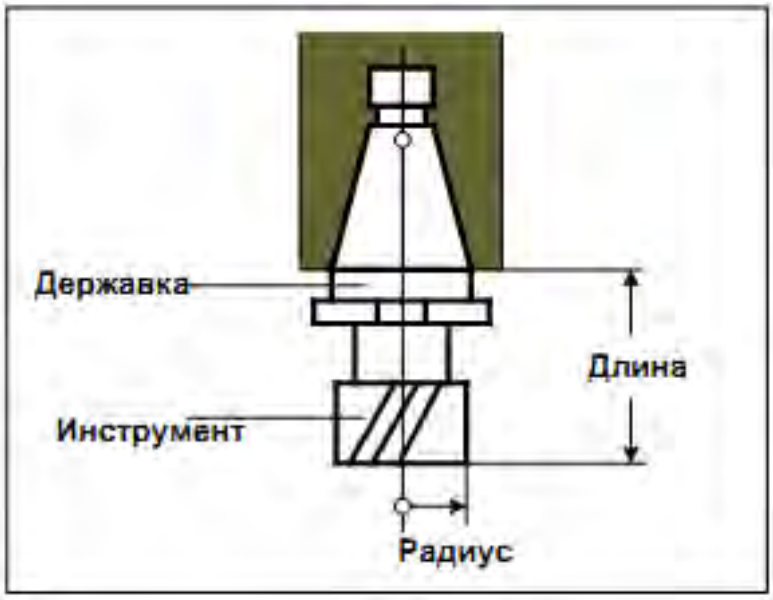



Рис.2. 10.

Компенсация длины возможна двумя способами: по отношению к передней плоскости шпинделя (см. рис.2.11) и по отношению к «нулевому инструменту» (см. рис.2.12).

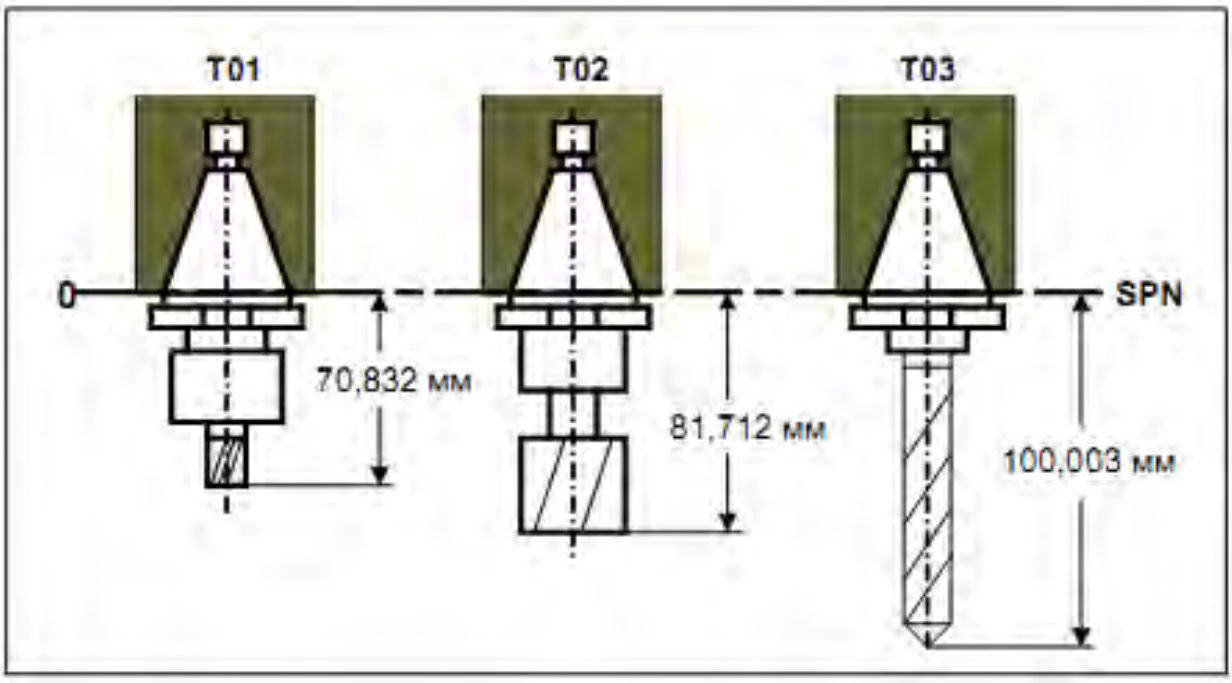



Рис. 2.11.

В обоих случаях величины компенсации сохраняются в соответствующей таблице. На рис. 11 для T01,- H1 =70.8320; для T02, - H2 = 81.7120; для T03, - H3 = 100.0030. Как видим, знак компенсации здесь может быть

только положительным.

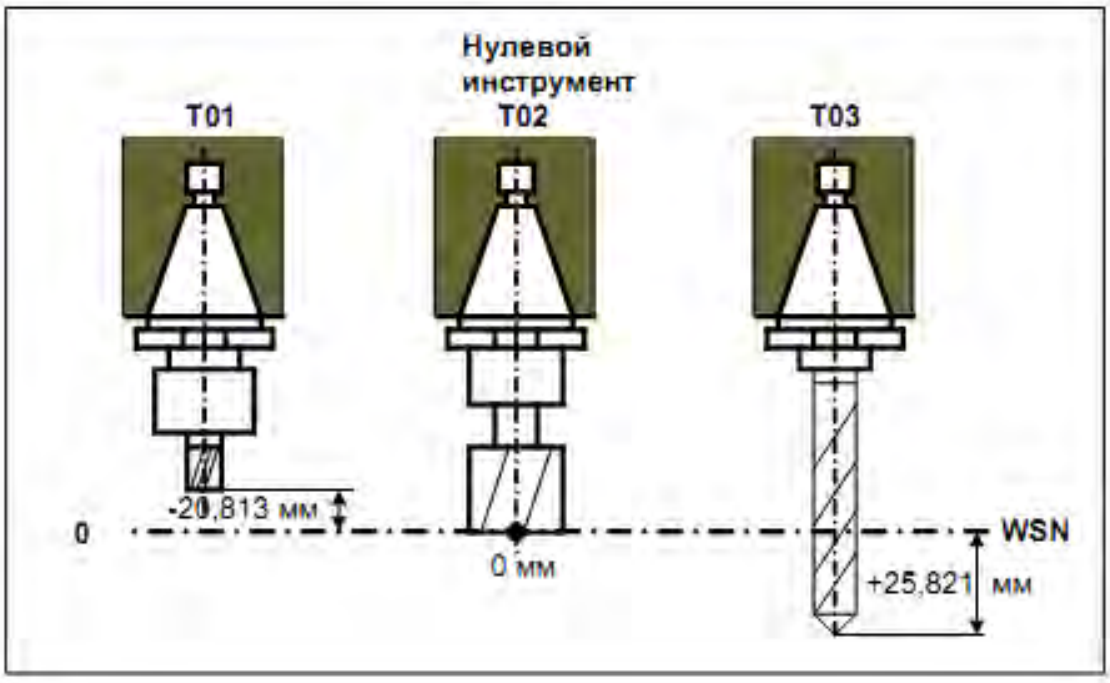



Рис. 2.12.

Во втором случае выбирают «нулевой инструмент», торцевая плоскость которого WSN (Workplane for Setting Null) служит для настройки и определения компенсации для всех остальных инструментов. «Нулевой инструмент» (T02 на рис.12) имеет нулевое значение компенсации. Знак компенсации может быть положительным или отрицательным. Например: для T01, - H1=-20. 813; для T02, - H2 = 0; для T03, - H3 = 25.821.

Центр фрезы движется по эквидистантной траектории, параллельной контуру детали, отстоящей от нее на величину, равную радиусу фрезы. Эквидистантную траекторию называют также траекторией центра фрезы. Значения компенсации для различных инструментов вносят в таблицу; например: для T01, - D1 = 14 (при диаметре фрезы 28 мм); для T02, - D2 = 22 (при диаметре фрезы 44 мм). Детали эквидистантной коррекции (компенсации) будут рассмотрены при анализе G-инструкций G40, G41 и G42.

Внешняя компенсация инициируется программируемым контроллером с помощью G- инструкций G145 и G845.

Так называемая «комплексная компенсация» представляет собой набор компенсационных данных для 3D-коррекции инструмента; или, например, для компенсации на длину в операциях с несколькими сверлами. Этот вид компенсации активизируется G-инструкциями G147 и G847. Комплексная компенсация может включать коррекцию на расположение режущей кромки.

**Траектории движения (типы интерполяции).** Линейная интерполяция предполагает движение по прямой линии в трех-координатном пространстве. Перед началом интерполяционных расчетов система ЧПУ

определяет длину пути на основе запрограммированных координат. В процессе движения осуществляется контроль контурной подачи так, чтобы ее величина не превышала допустимых значений. Движение по всем координатам должно завершиться одновременно.

При круговой интерполяции движение осуществляется по окружности в заданной рабочей плоскости. Параметры окружности (например, координаты конечной точки и ее центра) определяются до начала движения на основе запрограммированных координат. В процессе движения осуществляется контроль контурной подачи так, чтобы ее величина не превышала допустимых значений. Движение по всем координатам должно завершиться одновременно.

Винтовая интерполяция представляет собой комбинацию круговой и линейной. В процесс интерполяции вовлекаются синхронные координатные оси; например, X, Y и Z. Вспомогательные (асинхронные) координатные оси в процесс интерполяции не вовлекаются. Примером движения вдоль асинхронной оси может служить позиционирование инструментального магазина. Axes used as auxiliary axes (positioning of the tool magazine, e.g.) are called "asynchronous axes". При задании скорости подачи асинхронной оси используют адрес FA.

## 2.3 G-**инструкции**

**1. Линейная интерполяция при ускоренном перемещении**, - G00. Эффект состоит в том, что запрограммированное перемещение интерполируется, а движение к конечной точке осуществляется по прямой линии с максимальной подачей. Скорость и ускорение подачи, по крайней мере, одной оси максимальны. Скорость подачи других осей контролируется таким образом, чтобы движение всех осей завершилось в конечной точке одновременно. При активной инструкции G00 движение замедляется до нуля в каждом кадре. При этом выполнение «точного позиционирования» зависит от инструкций G161, G162. Если же в замедлении скорости подачи до нуля в каждом кадре необходимости нет, то вместо G00 используют G200. Значение максимальной скорости подачи не программируется, но задается так называемыми «машинными параметрами» в памяти системы ЧПУ. Инструкция G00 является модальной, и ее появление деактивирует G-инструкции той же группы: G01, G02, G03, G05, G10-G13, G73, G200.

**2. Линейная интерполяция на ускоренном перемещении без замедления до V=0,** - G200. Эффект состоит в том, что отсутствует замедление скорости подачи до нуля в конце каждого кадра; т.е. нет торможения на стыке соседних кадров, и процесс интерполяции продолжается. При этом должны соблюдаться предусловия: инструкции G61 и G163 пассивны. Если, тем не менее, инструкция G61 активна, то, несмотря на G200, торможение до нуля будет осуществляться в каждом кадре. Если же активна инструкция G163, то характер движения будет определяться функциями точного позиционирования (см. инструкции G164 - G166). Значение максимальной скорости подачи не программируют, но задают «машинными параметрами» в памяти системы ЧПУ. Инструкция G200 является модальной, и ее появление деактивирует G-инструкции той же группы: G00, G01, G02, G03, G05, G10-G13, G73.

**3. Линейная интерполяция с предусмотренной скоростью подачи**, - G01. Перемещение с заданной скорость подачи (в F-слове) к конечной точке кадра осуществляется по прямой линии. Все координатные оси завершают движение одновременно. Скорость подачи в конце кадра снижается до нуля, но только если инструкция G08 пассивна. Запрограммированная скорость подачи является контурной, т. е. значения подачи для каждой отдельной координатной оси будут меньше. Значение скорости подачи обычно ограничивают настройкой «машинных параметров». Вариант комбинации слов с инструкцией G01 в кадре: G01_ X_ Y_ Z_F_. Особенности использования инструкции G01:

- в любом кадре инструкция G01 может быть представлена вместе с позиционными данными или без них;

- в любом кадре инструкция G01 сопровождается F-словом, если до этого подача не была назначена;
- назначенная подача остается активной, пока ее значение не будет переопределено.
- инструкция G01 является модальной, и ее появление деактивирует G-инструкции той же группы: G0O, G02, G03, G05, G10-G13, G73, G200.

Фрагмент программы:

```
X100 Y100          /Начальное положение.
G01 X500 Y300 F100 /Движение к конечной точке.
```

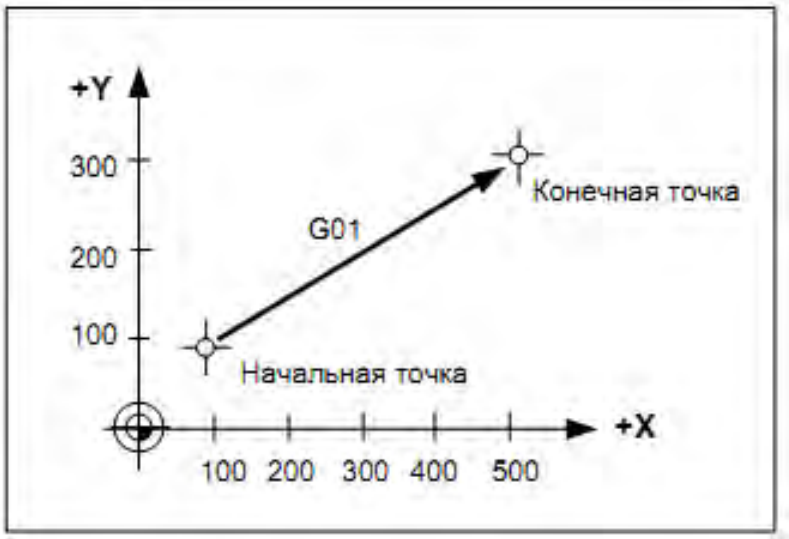



Рис.2.13.

**4. Круговая интерполяция,** - G02, G03. Перемещение в кадре осуществляется по окружности с контурной скоростью, заданной в активном F-слове. Движение по всем координатным осям завершается в кадре одновременно; также и в том случае, когда одна из осей не принадлежит плоскости круговой интерполяции. Вдоль этой оси движение будет линейно интерполируемым, а общая траектория станет винтовой линией. Инструкции G02 и G03 модальны и деактивируют другие G-инструкции той же группы. Приводы подачи задают перемещение по окружности с запрограммированной подачей в выбранной плоскости интерполяции; при этом G02 определяет движение по часовой стрелке, а G03 - против часовой стрелки. Выбор двух синхронных координатных осей осуществляется свободно путем выбора плоскости интерполяции.

При программировании окружность задают с помощью ее радиуса или координат ее центра. Дополнительная опция программирования окружности определяется инструкцией G05: круговая интерполяция с выходом на траекторию по касательной (см. далее).

**4.1 Программирование окружности при помощи радиуса.** Радиус всегда задают в относительных координатах; в отличие от конечной точки дуги, которая может быть задана как в относительных, так и в абсолютных координатах. Используя положение начальной и конечной точек, а также и значение радиуса, система ЧПУ прежде всего определяет координаты центра окружности. Результатом расчета могут стать координаты двух точек, ML MR (см. рис.2.14), расположенных соответственно слева и справа от прямой, соединяющей начальную и конечную точки.

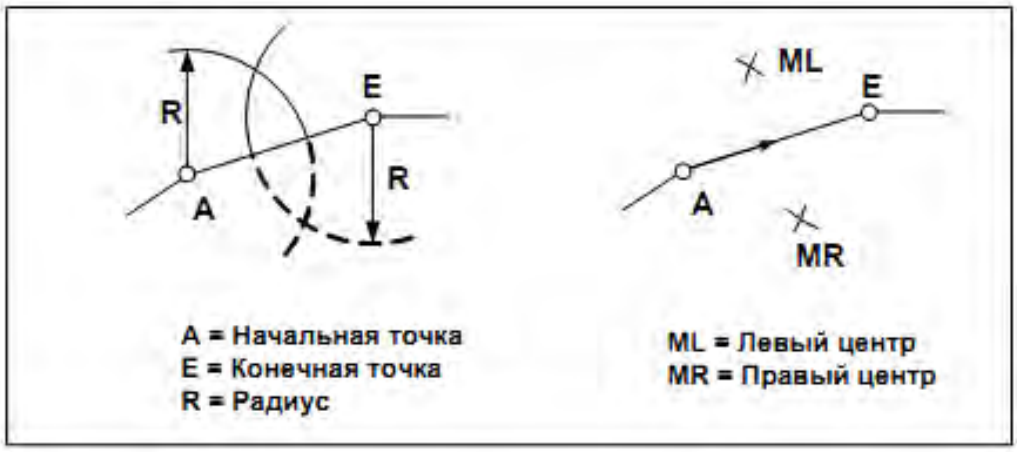



Рис.2. 14.

Расположение центра окружности зависит от знака радиуса; при положительном радиусе центр будет находиться слева, а при отрицательном радиусе – справа. Расположение центра определяется также инструкциями G02 или G03 (см. рис.2.15).

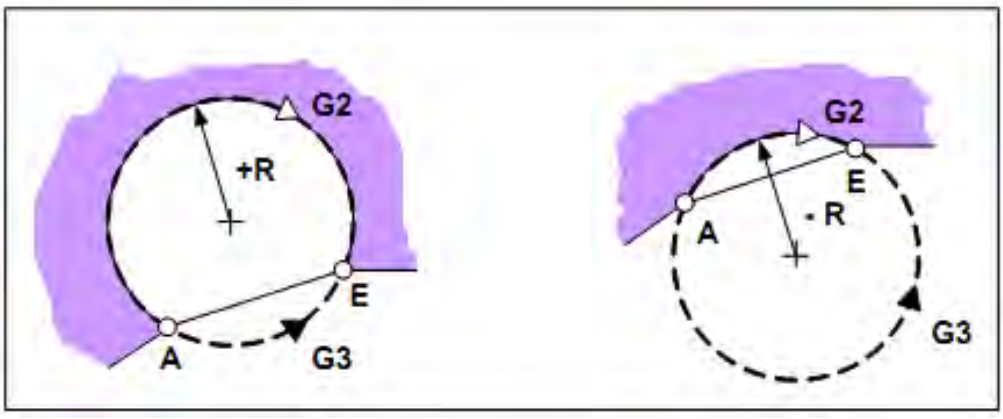



Рис.2. 15.

Как это видно из рис.2.15, величина радиуса должна быть, по крайней мере, вдвое большей, чем длина отрезка, соединяющего начальную и конечную точки дуги окружности. Особым случаем является равенство

отрезка удвоенному значению радиуса. Этот случай соответствует заданию полуокружности. Знак радиуса при этом значения не имеет. Программирование полной окружности через задание радиуса недопустимо. Вариант комбинации слов с инструкцией G03 в кадре: N_G17_G03_X_Y_R±_F_S_M. Здесь: инструкция G17 означает выбор круговой интерполяции в плоскости X/Y; инструкция G03 определяет круговую интерполяцию в направлении против часовой стрелки; X_Y_ представляют собой координаты конечной точки дуги окружности; R - радиус окружности.

4.2. **Программирование окружности при помощи координат ее центра**. Текущее положение используется в качестве начальной точки. Окружность, заданная координатами центра, проходит через начальную и конечную ее точки. Координатные оси, вовлеченные в процесс круговой интерполяции, имеют параметры I, J и K, приданные осям соответственно. Параметры устанавливают расстояние между начальной точкой и центром M дуги окружности в направлении, параллельном осям. Знак определяется направлением вектора от A к M. Стандартное определение параметров указано на рис.2.16.

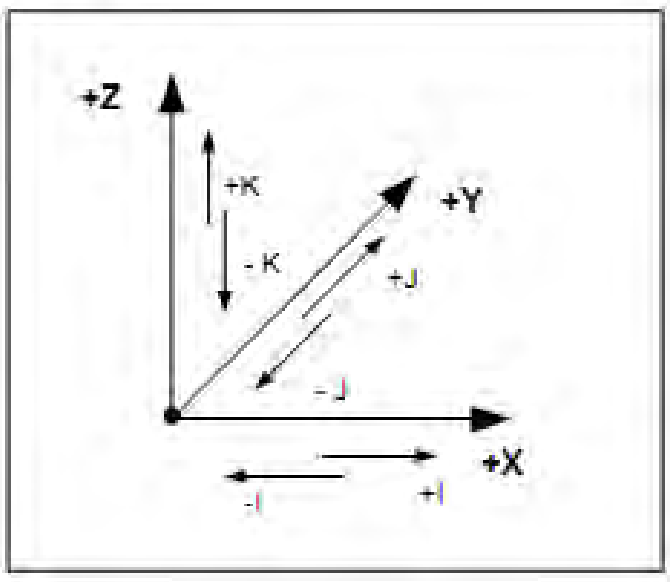



Рис.2. 16.

На рис.16: I=M(X)-A(X); J =M(Y)-A(Y); K=M(Z)-A(Z); I, J, K – параметры интерполяции; X, Y, Z – координатные оси, которым параметры I, J, K приданы соответственно; M – центр окружности, заданный относительно начальной точки дуги окружности.

На рис. 2.17-2.21 рассмотрены различные примеры программирования окружности.

**Пример** 1:

```
N…G90 G17 G02 X350 Y25O I200 J-50 F…S…M…
```

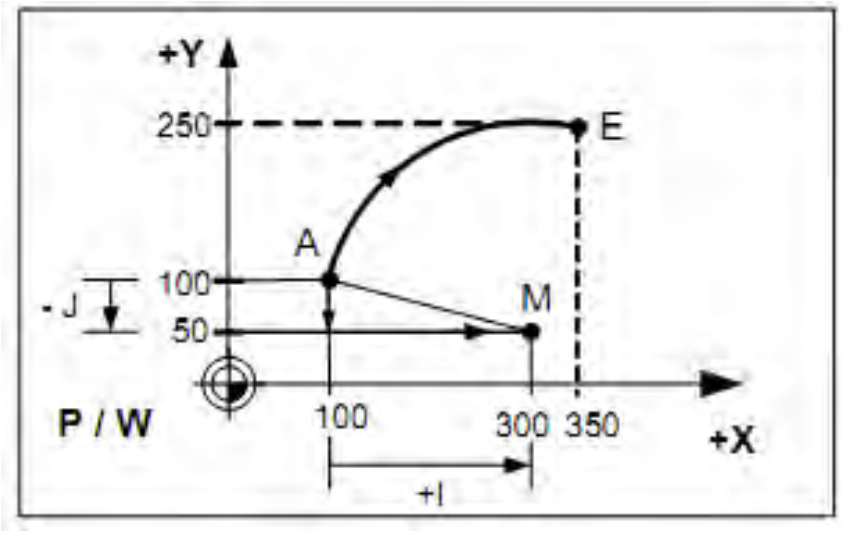



Рис. 2.17.

**Пример 2:**

```
N…G90 G17 G03 X350 Y200 I-50 J200 F…S…M…
```

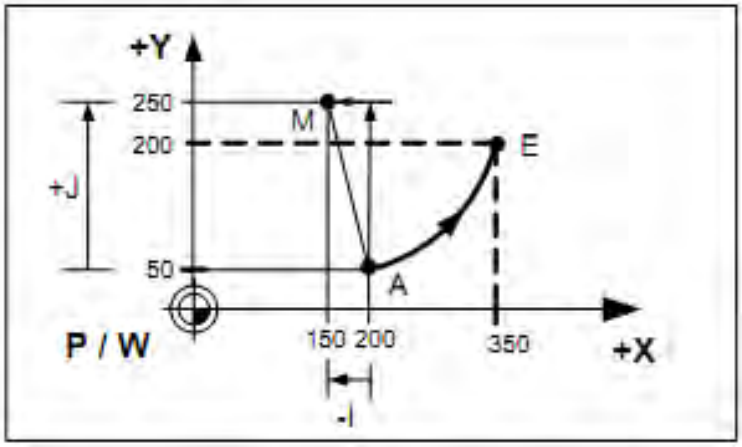



Рис. 2.18.

**Пример 3 (программирование четверти окружности)**:

```
N…G17 G02 X…Y…J-…F…S…M…
```


Рис.2. 19.

**Пример 4 (программирование полуокружности):**

N…G17 G03 X…I…F…S…M…


Рис.2. 20.

**Пример 5 (программирование полной окружности):**

N…G17…G02 I…F…S…M…


Рис.2. 21.

**5. Винтовая N-интерполяция,** - G202, G203. В процессе винтовой N-интерполяции осуществляется круговая интерполяция в выбранной плоскости и линейная интерполяция для остальных синхронных координатных осей, общим числом до шести круговых или линейных осей. Это связано с тем, что общее число синхронных осей в одном канале не превышает восьми. Движение по всем координатам завершается одновременно. Винтовая N-интерполяция является обобщением простой винтовой, при которой линейная интерполяция осуществляется только для одной оси, перпендикулярной выбранной плоскости круговой интерполяции.

Плоскость круговой интерполяции определяется инструкциями G17, G18, G19, G20. В одном кадре может быть запрограммирована только одна полная окружность. Скорость подачи является контурной; однако есть некоторые особенности для линейно интерполируемых осей, связанные с использованием инструкций G594 и G595.

Движение по окружности по часовой стрелке осуществляется соответственно инструкции G202; движение по окружности против часовой стрелки осуществляется соответственно инструкции circular G203. Программирование окружности возможно с использованием радиуса и координат центра окружности.

Инструкция винтовой интерполяции является модальной и принадлежит второй группе модальных G-инструкций.

Пример простой винтовой интерполяции показан на рис.2. 22:

```
N…G91 G17 G03 X…Y…Z…I…J…F…S…M…
```

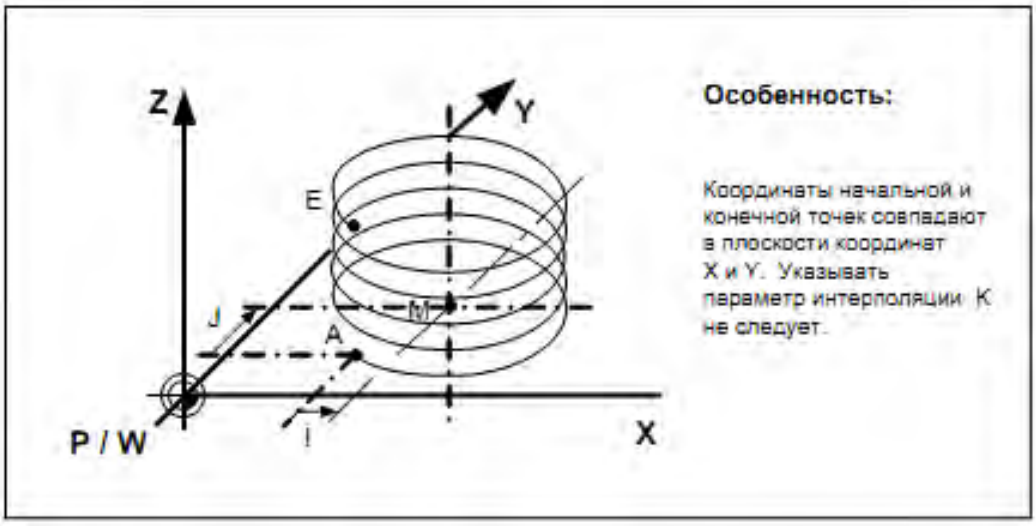



Рис.2. 22.

**6. Выдержка времени,** - G04. Инструкция G04 указывает на сам факт выдержки времени, а в слове F задают величину этой выдержки в секундах. Действие инструкции распространяется только на один кадр. В этом же кадре можно программировать вспомогательные функции (например, смену инструмента), но не перемещения. Движение приводов подачи останавливается, а вращение шпинделя не выключается.

Пример программирования выдержки времени:

```
N… G04 F…          /Выдержка времени в секундах.
```

Повторную выдержку времени следует программировать в очередном кадре.

**7. Круговая (винтовая) интерполяция с выходом на круговую траекторию по касательной,** - G05. Система ЧПУ использует инструкцию G05 для расчета такого кругового участка, выход на который из предыдущего кадра (с линейной или круговой интерполяцией) осуществляется по касательной. Параметры формируемой дуги определяются автоматически; т.е. программируется только ее конечная точка, а радиус не задается: G5 X…Y... Различные примеры программирования с инструкцией G05 показаны на рис.2.23.

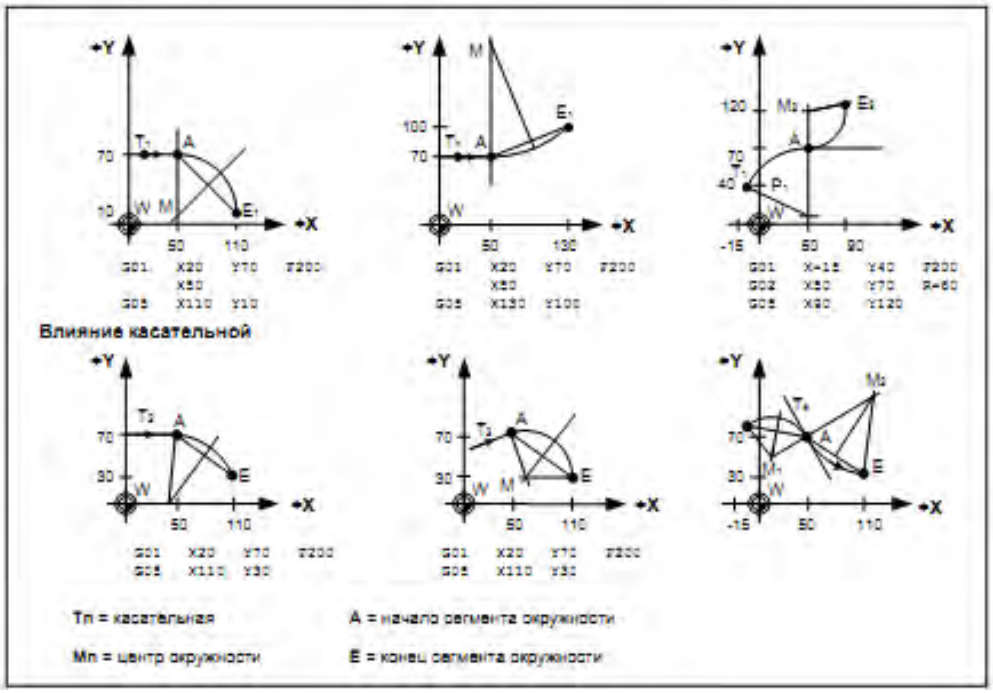



Рис.2.23.

**8. Программирование ускорения,** - G06, G07, G206. Максимальные значения ускорения по каждой координатной оси устанавливают в «машинных параметрах». Временно эти значения могут быть снижены инструкцией G06.

Инструкция G06, сопровождаемая адресами осей, заменяет для этих осей максимальные значения ускорения на те, которые указаны в функциях осей. Эти значения интерпретируются системой управления как «тысячи Дюймов/Сек$^2$» или как «М/Сек$^2$», в зависимости от выбора единиц измерения инструкциями G71, G70 соответственно. Желательно программировать инструкцию G06 в отдельном кадре.

Инструкция G07 отменяет введенные изменения для всех осей сразу, т.е. значения соответствующих «машинных параметров восстанавливаются. Инструкцию G07 можно применять одновременно с программированием перемещений.

Инструкцией G206 сохраняют во внутренней памяти системы ЧПУ действующие максимальные значения ускорения для всех координатных осей. Инструкция G06, не сопровождаемая адресами осей, вновь активизирует максимальные значения ускорения, сохраненные во внутренней памяти системы ЧПУ. Использование инструкций поясняется двумя примерами.

**Пример 1:**

```
G06 X2 Y2   /Максимальное значение ускорения для осей X и Y равно
            /2М/Сек².
```

**Пример 2**: первоначально максимальное ускорение 8М/Сек$^2$ задано в машинных параметрах для всех координатных осей.

```
G06 X1.0 Z2.1     /Максимальное ускорение для оси X равно 1.0 М/Сек²;
...               /максимальное ускорение для оси Y равно 8.0 М/Сек²;
                  /максимальное ускорение для оси Z равно 2.1 М/Сек².
G206      /Запоминание текущих максимальных значений ускорения.
...
G07       /Активизация значений, сохраненных в машинных  параметрах:
...          /максимальное ускорение для оси X равно 8.0 М/Сек²;
             /максимальное ускорение для оси Y равно 8.0 М/Сек²;
             /максимальное ускорение для оси Z равно 8.0 М/Сек².
G06 Y5       /Максимальное ускорение для оси X равно 8.0 М/Сек²;
...          /максимальное ускорение для оси Y равно 5.0 М/Сек²;
             /максимальное ускорение для оси Z равно 8.0 М/Сек².

G06          /Максимальное ускорение для оси X равно 1.0 М/Сек²;
...          /максимальное ускорение для оси Y равно 8.0 М/Сек²;
             /максимальное ускорение для оси Z равно 2.1 М/Сек².
```

**9. Управление скоростью подачи в «точках перегиба» траектории,** - G08, G09. Указанные инструкции поддерживают контурную скорость подачи вдоль сложной траектории настолько постоянной, насколько это возможно. Если подобное управление выключено, то скорость подачи снижается до нуля в конце каждого кадра и возрастает до запрограммированного значения в начале каждого кадра. Если подобное управление включено, то скорость подачи будет снижаться до необходимого уровня в точках перегиба контура (за исключением начала и конца процесса обработки), см. рис.2.24.

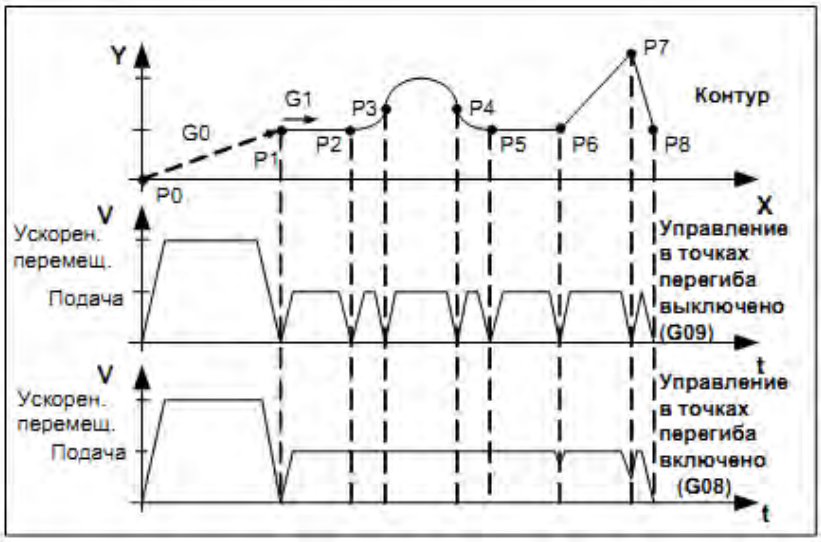



Рис. 2.24.

При активной инструкции G08 (управление в точках перегиба включено) конечная точка P8 будет достигнута за более короткое время, чем при активной инструкции G09 (управление в точках перегиба выключено). Обе инструкции модальны.

**Пример.**

```
N…G08                 /Управление в точках перегиба включено.
N…G0O X100 Y50        /Ускоренное перемещение к точке P1.
N… G01 X150 F5000     /Продолжение движения со скоростью подачи.
N…
```

**10. Управление скоростью подачи в точках перегиба с учетом функции "look-ahead",** - G108. Функция "look-ahead" работает в диапазоне числа кадров, установленных в машинных параметрах. Завершение диапазона для инструкции G08 выглядит так же, как и завершение процесса обработки. В этом случае произойдет неоправданное снижение скорости, как это было показано на рис. 24. Инструкция G108 позволяет избежать подобного снижения скорости подачи.

**Пример.**

```
G08
…
```

```
N998 X1000
N999 G0O X5000 G108   /В конце кадра скорость подачи снизится лишь
                      /до необходимого уровня.
N1000 X5001
M30
```

**11. Переходы от кадра к кадру без торможения,** - G228. Алгоритм управления скоростью подачи в точках перегиба на стыке кадров принимает во внимание величину скорости подачи, максимально допустимый скачок скорости и допустимую длину участка торможения. При этом торможение может произойти даже при незначительных изменениях угла наклона контура, т.е. при квази-гладкой траектории. Инструкция G228 позволяет установить угол излома контура, в пределах которого торможение не происходит. Инструкция программируется следующим образом: G228 {K<угол излома контура>}; где К - адрес функции, в которой указывают максимальный угол в пределах от 0 до 50 градусов.

**12. Формирование «гладкого» ускорения при движении от точки к точке,** - G408. Цель состоит в растягивании процедуры ускорения на величину нескольких интерполяционных циклов без скачков скорости подачи. В качестве закона изменения скорости подачи принимают, например, функцию $\sin^2$.

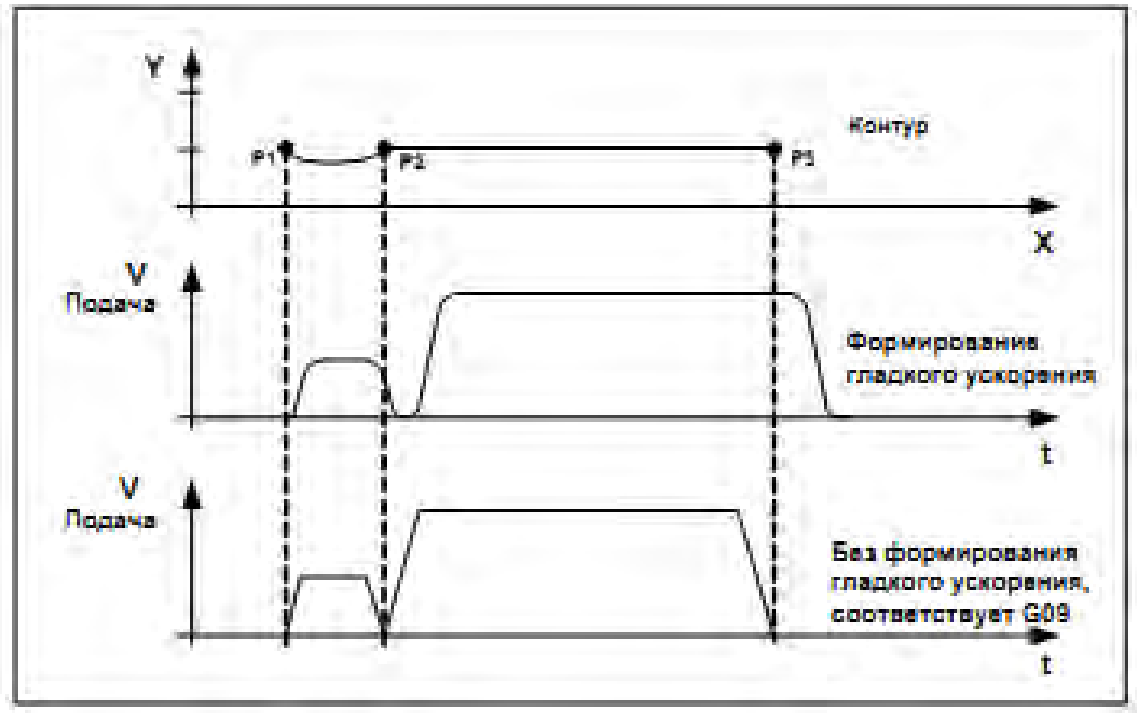



Рис.2. 25.

Параметрами инструкции служат LIN и SIN. Параметр LIN <число> служит признаком линейного изменения ускорения; причем <число> означает количество интерполяционных циклов (от 2 до 40), между которыми распределяется линейное изменение ускорение. Параметр SIN <числовое значение> служит признаком изменения ускорения соответственно функции $\sin^2$; а <числовое значение> является вариантом использования инструкции G408 (см. рис.2.26):

- SIN 0, формирование ускорения отменяется (эквивалентноG09);
- SIN 3, изменение ускорение охватывает три интерполяционных цикла в отношении 25% - 50% - 25%;
- SIN 4, изменение ускорение охватывает четыре интерполяционных цикла в отношении 12.5% - 37.5% -37.5% - 12.5%;
- SIN 5, изменение ускорение охватывает пять интерполяционных циклов;
- SIN 10, изменение ускорение охватывает 10 интерполяционных циклов;
- SIN 15, изменение ускорение охватывает 15 интерполяционных циклов;
- SIN 20, изменение ускорение охватывает 20 интерполяционных циклов;
- SIN 40, изменение ускорение охватывает 40 интерполяционных циклов; Параметр SIN имеет более высокий приоритет, чем параметр LIN.

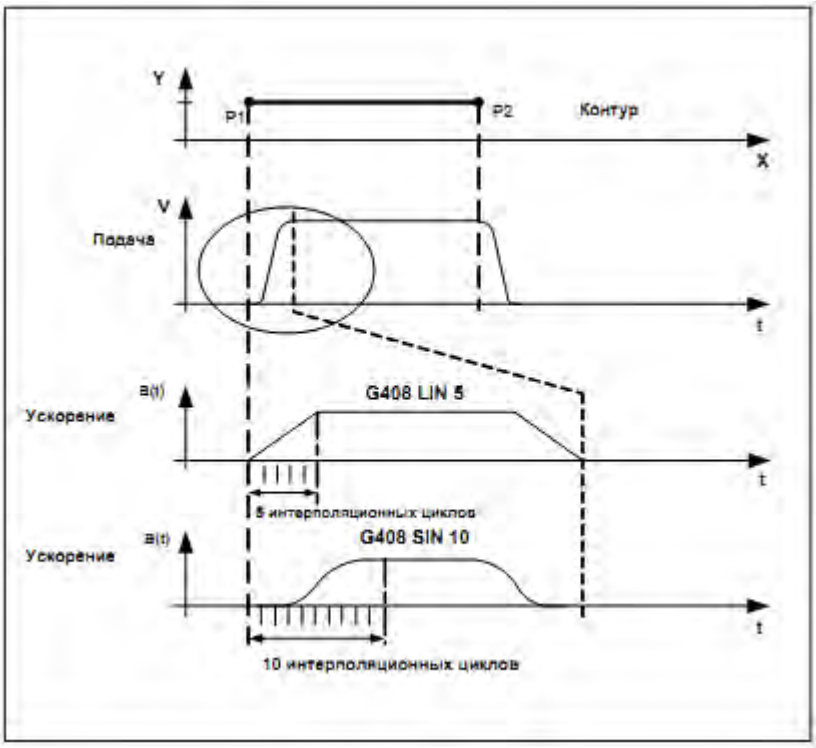



Рис.2. 26.

Примеры корректного использования инструкции G408:

- G408; по умолчанию используется G408 LIN 2.
- G408 SIN 3 LIN 5; изменение ускорения соответствует SIN 3;
- G408 LIN 5; изменение ускорения соответствует LIN 5;
- G408 LIN 2; изменение ускорения соответствует LIN 2.

Инструкция G408 модальная (принадлежит группе инструкций G08, G09, G108, G608).

**13. Формирование «гладкого» ускорения при движении от точки к точке для каждой оси в отдельности,** - G608. Ускорение формируется для каждой синхронной оси независимо. В процессе интерполяции система управления автоматически определяет общую функцию формирования ускорения.

Синтаксис инструкции: G608 <ось i> < число интерполяционных циклов для формирования ускорения оси i>…<ось n> <число интерполяционных циклов для формирования ускорения оси n>.

**Пример.**

```
N...G608 X4 Y6 Z10   /Число интерполяционных циклов для оси X
                     /равно 4, для оси Y равно 6, для оси Z равно 10.
```

Примечание:

- инструкции G608, G08 G09 G408 являются модальными и прекращают действие других из той же группы;
- число интерполяционных циклов может быть назначено от 1 до20; для неупомянутых осей принимается число интерполяционных циклов, заданное
- в машинных параметрах;
- если инструкция G608 не сопровождается перечислением осей, то для них всех
- принимается число интерполяционных циклов, заданное в машинных параметрах;
- инструкция G608 предполагает торможение до нуля и в этой связи используется при позиционировании.
- при инициализации системы активна инструкция G09.

**14. Программирование в полярных координатах, - от G10 до G13.**

При программировании в полярных координатах положение точки определяется через радиус и углом. Полюс и плоскость координат задают с помощью инструкции G20.

**Пример:**

```
G20 X100 Z100     /Программирование осуществляется в плоскости
                  /X_Z, а декартовы координаты полюса равны X =
                  /100, Z = 100.
```

Примечание.

- Если координаты полюса не указаны, то полюс совпадает с началом декартовых координат.
- Положение точки задают начальным положением радиуса (совпадающим с одной из двух осей, определяющих плоскость полярных координат), величинами радиуса и угла. Угол отсчитывается по отношению к начальному радиусу. Функция A может иметь различный синтаксис, который устанавливается машинными параметрами.

**Пример 1** (см. Рис.2.27).

```
N150 G20 Z25 X10    /Задание положения полюса.
N160 G10 Z20 A70    /Задание начального положения радиуса,
                    /величин радиуса и угла.
```

**Пример 2** (см. Рис.2.27).

```
N150 G20 Z30 X20    /Задание положения полюса.
N160 G10 X20 A70    /Задание начального положения радиуса,
                    /величин радиуса и угла.
```

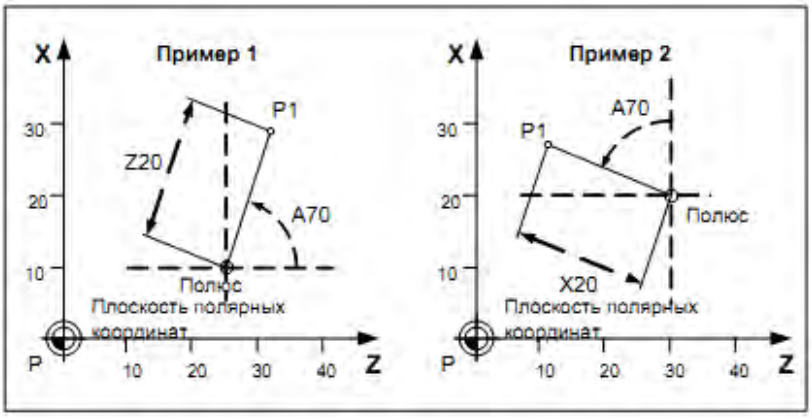



Рис.2. 27.

Инструкции программирования:
G10, - ускоренное перемещение в полярных координатах (по типу G0O).
G11, - линейная интерполяция в полярных координатах (по типу G01). 33
G12, - круговая интерполяция по часовой стрелке в полярных координатах (по типу G02).
G13, - круговая интерполяция против часовой стрелки в полярных координатах (по типу G03).
Инструкции G0O, G01, G02, G03, G05, G10-G13 являются модальными и отменяют одна другую.

**15. Инструкции программирования коэффициента KV усиления по скорости следящего привода подачи,** - G14, G15. Инструкции позволяю программно изменять коэффициент KV для каждой отдельной координатной оси. Обычно этим пользуются для кратковременного повышения «жесткости» следящего привода в пределах некоторых технологических операций. В остальных случаях значение коэффициента устанавливается в машинных параметрах. В тех кадрах, которые предшествуют программному изменению значения коэффициента, должно осуществляться торможение до нулевой скорости подачи; поскольку изменять коэффициент можно лишь в статическом состоянии привода. Область использования – кратковременное изменение жесткости привода.

Значение коэффициента: KV = V/S; где V М/Мин - есть скорость подачи, а S – есть ошибка следящего привода по скорости. Инструкция G14 открывает возможность программирования коэффициента KV; а инструкция G15 закрывает такую возможность.

**Пример:**

```
G14 X1.20 Y1.20 Z1.20    /Значение KV устанавливается равным 1.2
                         /для осей  X,  Y и Z .
G14 Z1.4                 /Значение KV устанавливается равным 1.4
                         /для оси Z .
G15 X200 Y300 Z-150      /Возвращается то значение KV, которое
                         /было установлено машинными
                         /параметрами.
```

Максимальное значение коэффициента KV равно 655. 35. Инструкцию G15 можно использовать без позиционной информации в кадре.

**16. Программирование без указания плоскости,** - G16. Инструкция G16 используется в следующих случаях.

- Если одна из осей, образующих заданную ранее плоскость, удаляется из канала; то инструкция G16 активизируется автоматически. Круговая интерполяция невозможна вплоть до выбора новой плоскости интерполяции.
- Если ни одна из инструкций выбора плоскости (G17, G18, G19, G20) не действует по умолчанию после инициализации системы, то инструкция G16 активизируется автоматически.
- Если ни в круговой, ни в винтовой интерполяции нет необходимости (например, если каналу приписана только одна координатная ось).

Инструкция G16 деактивирует выбор плоскости. Эта инструкция является модальной и действует в той же группе, что и G17, G18, G19, G20.

**17. Выбор плоскости, - G17 (плоскость X_Y), G18 (плоскость Z_X), G19 (плоскость Y_Z).** Инструкции определяют выбор рабочей плоскости в системе координат детали или программы. Работа инструкций G02 G03 G05 непосредственно связана с этим выбором; так же, как и программирование в полярных координатах; так же, как и эквидистантная коррекция (см. рис.2.28).

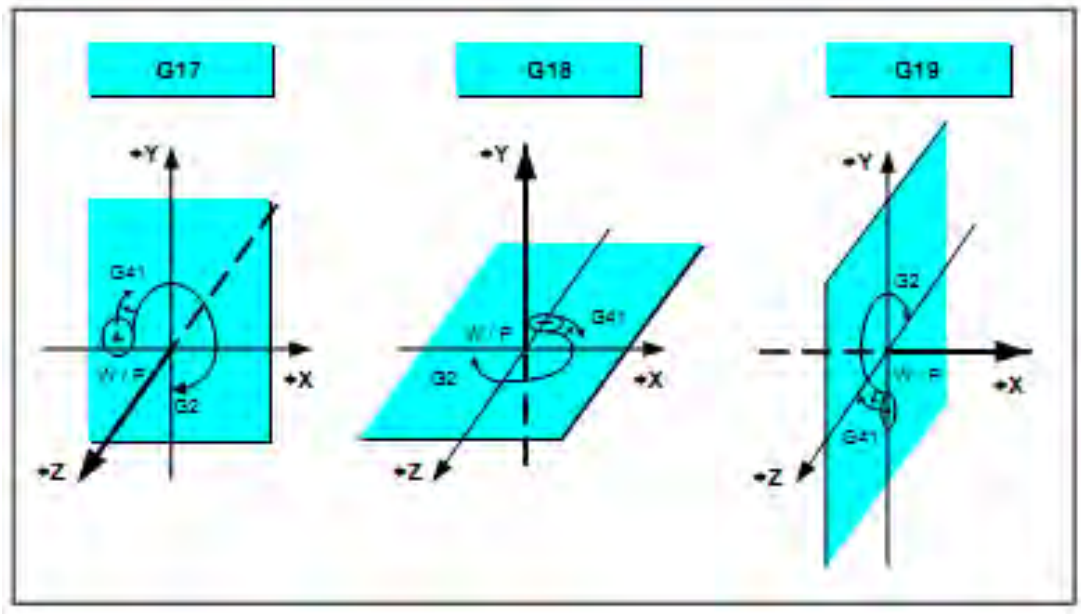



Рис.2.28.

В современных системах ЧПУ наименование осей может быть иным, нежели X, Y и Z. В этом случае выбор плоскости и назначение параметров интерполяции осуществляют в соответствии с классификацией осей (зафиксированной в машинных параметрах) согласно следующей схеме:

| | Классификационный номер | Классификационный номер |
|---|---|---|
| | главной оси и приданный ей параметр интерполяции | вторичной оси и приданный ей параметр интерполяции |
| **G17** | **1 (I)** | **2 (J)** |
| **G18** | **3 (K)** | **1 (I)** |
| **G19** | **2 (J)** | **3 (K)** |

**18. Свободный выбор плоскости интерполяции для двух осей, назначение полюса для программирования в полярных координатах, -** G20. В кадре с инструкцией G20 задают координатные оси, определяющие плоскость интерполяции и коррекции на радиус фрезы. Если адреса осей сопровождаются числовой информацией, то система управления интерпретирует эту информацию как координаты полюса полярной системы координат (см. G10-G13).

**Пример:**

```
N...G20 XO YO        /Выбор плоскости X_Y в качестве плоскости
                     /интерполяции. Координаты полюса: X=0,Y=0).
N...G20 Y100 Z200    / Выбор плоскости X_Y в качестве плоскости
                     /интерполяции. Координаты полюса: Y=100, Z=200.
```

**19. Программирование классификации осей,** - G21. Классификация устанавливает:

- выбор осей, соответствующих инструкциям G17, G18 и G19; а также выбор признаков главной и вторичной осей;
- выбор признаков главной и вторичной осей в кадре с инструкцией G20;
- связывание параметров интерполяции I, J и K с осями в соответствии с установленной классификацией.

**Пример:**

```
N100 G17 XO YO ZO   /Классификация осей по умолчанию: X=1; Y=2; Z=3.
N200 G512 (Y)       /Y удаляется из составленной группы
                    /осей. Подразумевается переключение на G16.
                    /Круговая интерполяция становится невозможной.
N210 G511 (YA)      /Ось YA входит в группу осей без
                    /классификационного номера.
```

```
N220 G21 YA2        /YA получает классификационный номер 2.
N230 G17            /Активизация плоскости X_YA.
N240 G2 X YA        /Круговая интерполяция вновь возможна.
```

**20. Активизация таблиц,** - G22. Инструкцию G22 используют для активизации:

- таблиц смещения нуля;
- таблиц коррекции инструмента;
- таблиц, определяющих положение наклонных плоскостей.

Таблицы сохраняются в виде ASCII-файлов в файловой системе. Число таблиц ограничено объемом памяти файловой системы.

**Программирование:**

```
N...G22 V |<путь>|<имя файла>     /Активизация таблицы смещения нуля.
N...G22 K |<путь>|<имя файла>     /Активизация таблицы коррекции
                                  /инструмента.
N...G22 ID |<путь>|<имя файла>    /Активизация таблицы, определяющей
                                  /положение наклонной плоскости.
```

Здесь: <имя файла> - свободно выбираемое имя файла; <путь> - путь доступа к файлу.

**21. Программирование переходов: безусловного перехода G24 <номер кадра>; условного перехода G23 <интерфейсный сигнал>; перехода назад GOTOB; перехода вперед GOTOF.** Обычно кадры программ, подпрограмм и циклов выполняются в той же последовательности, в какой они запрограммированы. Однако эта последовательность может быть изменена при помощи переходов. Возможны различные варианты таких переходов.

**22. Нарезание резьбы без компенсирующего патрона,** - G32. Система ЧПУ осуществляет линейную интерполяцию между перемещением вдоль оси нарезания резьбы и вращением шпинделя, т.е. синхронизирует эти движения. При этом отпадает необходимость в компенсирующем патроне, сглаживающимнесоответствие между указанными перемещениями (при нарезании резьбы метчиком). Синтаксис инструкции выглядит следующим образом: G32 <ось нарезания резьбы> F<подача> M<3|4> S <частота вращения шпинделя> | H < шаг резьбы >. Задание частоты вращения шпинделя альтернативно заданию шага резьбы. (M3/M4) – признак прямого или реверсивного движения.

**Пример 1:**

```
N10 G0O  X20 Y15 Z10 F1000 S5000   /Позиционирование вдоль оси.
N20 G32 Z-20  F1000 M3 S1000       /Нарезание резьбы вдоль оси Z.
N30 G32 Z5  F1000 M4 S1000         /Реверсивное движение вдоль оси Z.
```

**Пример 2:**

```
N10 GO X30 Y5  ZO F1500        /Позиционирование вдоль оси.
N20 G32 Z-20  M3 H. 75         /Нарезание резьбы вдоль оси Z.
N30 G32  ZO M4 H. 75           /Реверсивное движение вдоль оси Z.
```

Нарезание резьбы и реверсивное движение должны осуществляться во втором примере с одним и тем же шагом.

**23. Сглаживание сопряжения кадров,** - G34, G35, G36, G134. Инструкции G34 и G134 задают скругление на стыке двух кадров с прямолинейными участками, а инструкция G134 выполняет ту же функцию на стыке кадров с круговыми или винтовыми траекториями. В результате выравнивается скорость подачи, и соблюдаются ограничения на ускорение. С другой же стороны, в процессе интерполяции поддерживается минимальное отклонение от запрограммированного и скорректированного контуров. Параметры отклонения устанавливаются в машинных параметрах, но могут и быть изменены в управляющей программе.

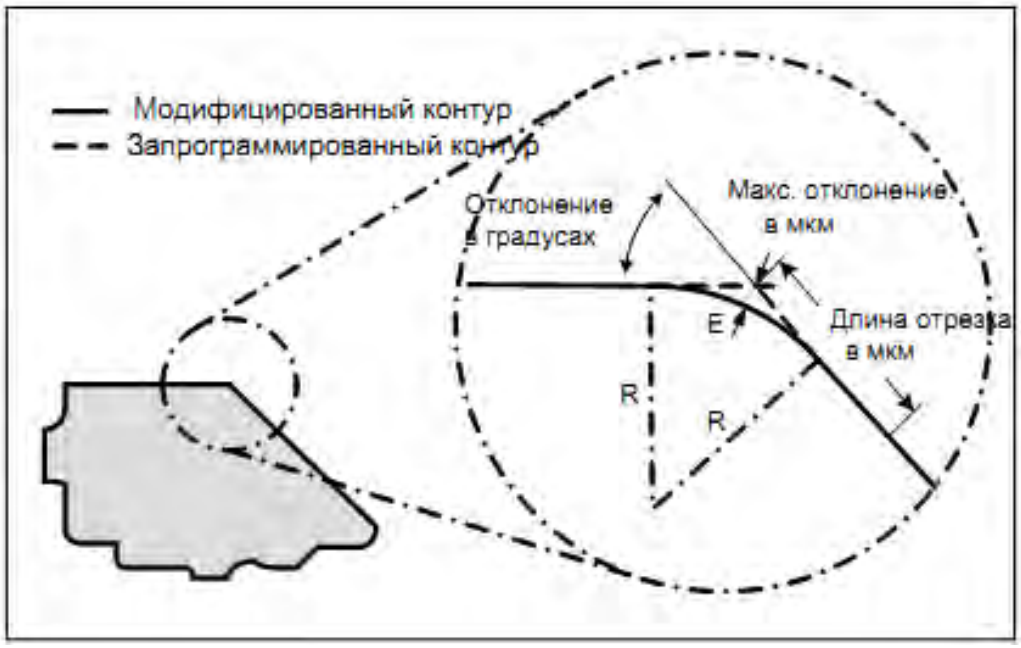



Рис.2. 29.

Инструкция G34 включает функцию скругления для двух линейных участков, а инструкция G35 выключает эту функцию. Для программирования отклонения скорректированного контура используют E-слово; это возможно только при активной инструкции G34. Инструкция G36 восстанавливает параметры отклонения, установленные в машинных параметрах. Инструкция G134 включает функцию скругления для двух круговых или винтовых участков. Для программирования радиуса скругления используют модальное R-слово. Задание радиуса возможно лишь при активной инструкции G134.

**24. Зеркальное отображение, масштабирование, поворот,** - G37, G38, G39. При зеркальном отображении, масштабировании и повороте отсутствует необходимость в изменение контура в исходной управляющей программе.

Можно использовать любую комбинацию этих функций (см. рис.2. 30).

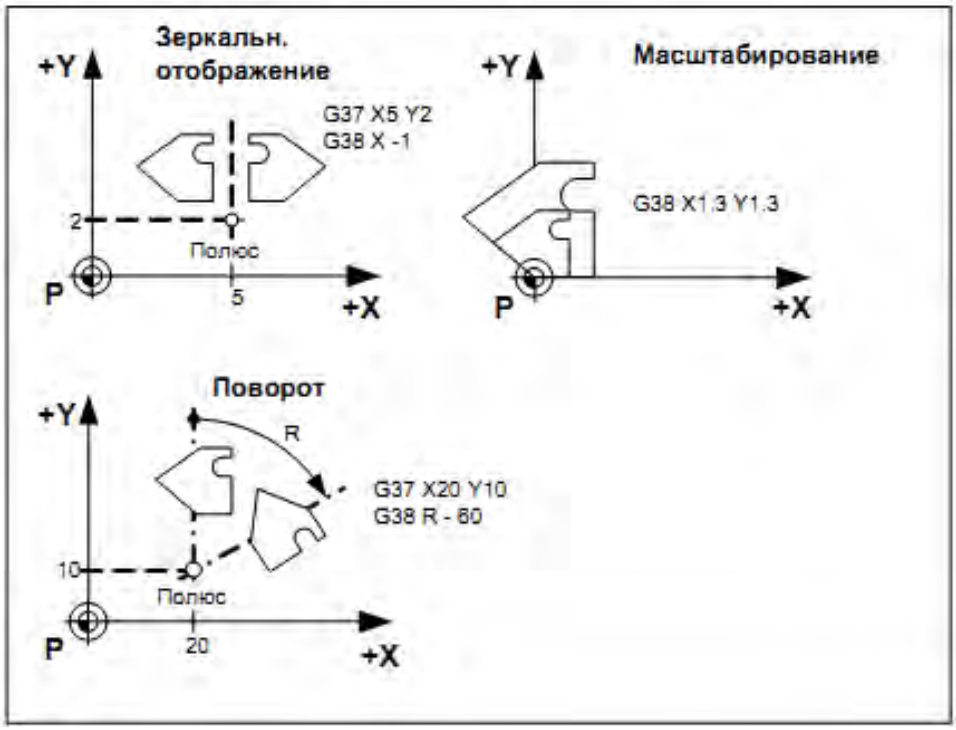
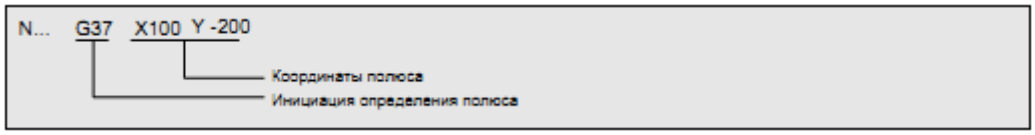



Рис.2. 30.

С помощью инструкции G37задают координаты точки, относительно которой осуществляется зеркальное отображение или поворот. С помощью инструкции G38 активизируют функции зеркального отображения, поворота, масштабирования. С помощью инструкции G39 активизируют функции зеркального отображения, поворота, масштабирования.

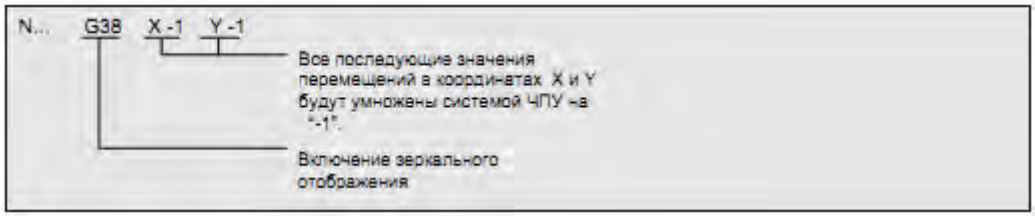
**24.1. Зеркальное отображение,** - G37, G38, G39. Модальная инструкция G37 сопровождается заданием абсолютных координат полюса (точки зеркального отображения) относительно нуля управляющей программы. Инструкцию можно использовать только в комбинации с G38.

**Пример:**




Модальная инструкция G38 включения зеркального отображения сопровождается адресом координатной оси и значением "-1". При этом вся позиционная информация для этой оси приобретает противоположный знак. При любом ином значении, отличающемся от единицы, будет осуществлено масштабирование.

**Пример:**




Модальная инструкция G39 выключает функцию зеркального отображения, уничтожая все оси такого отображения. Инструкция прекращает действие инструкций G37 и G38; и сбрасывает координаты полюса в нуль.

Примеры зеркального отображения представлены на рис.2.31.

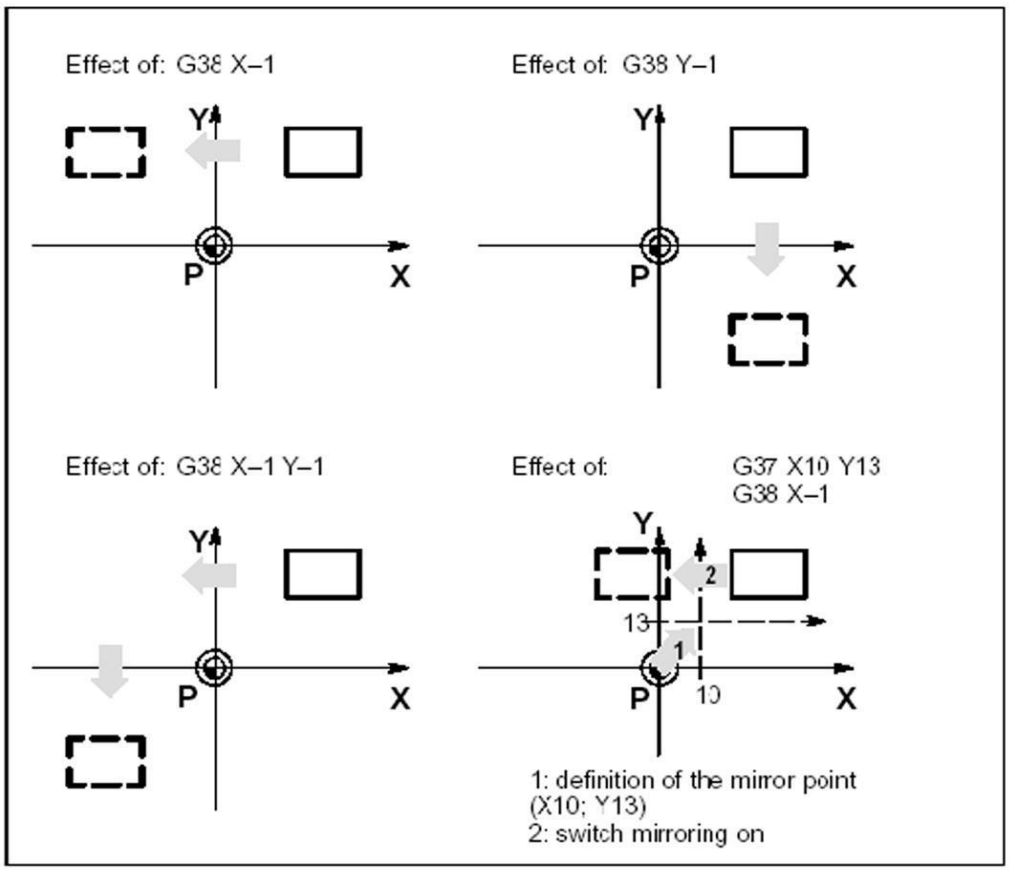



Рис.2. 31.

24.**2. Масштабирование,** - G38, G39. При масштабировании эталонный контур увеличивают или уменьшают. В особенности это удобно при использовании подпрограмм, когда перед их вызовом в основную программу вносят, если это нужно, коэффициент масштабирования. Это позволяет оставлять основную программу неизменной. Масштабирование не изменяет скорости подачи, а вспомогательные функции M02 и M30 в подпрограммах не выключают функции масштабирования. Коэффициент масштабирования устанавливают независимо для всех координатных осей; однако при круговой и винтовой интерполяции этот коэффициент должен быть для всех осей одинаковым. Коэффициент масштабирования изменяет параметры интерполяции I, J, K, R.

Инструкция масштабирования может работать вместе с инструкциями G0O, G01, G02, G03, G05, G10, G11, G12, G13, G20, G73, G90, G91, G190, G191, G200. Для инструкции G37 координаты полюса не меняются. Инструкция масштабирования не оказывает влияния на параметры коррекции инструмента, т.е. на инструкции G40, G41, G42, G43, G44. Инструкция масштабирования не оказывает влияния на координаты смещения нуля, т.е. на инструкции G54-G59, G154-G159, G254-G259. Программируемые смещения контура в соответствии с инструкцией G60 и компенсации в соответствии с инструкцией G92 не масштабируются. Масштабирование не связано с измерениями для инструкций G70, G71. Инструкция масштабирования становится пассивной при активных инструкциях G74, G76. Если фактор масштабирования оказывает влияние на координаты начальной точки контура, следует соответствующим образом запрограммировать нуль координатной системы детали.

Модальная инструкция G38 (см. рис.2.32) включает масштабирование для тех осей, которые указаны в кадре с положительным коэффициентом масштабирования. При этом все запрограммированные размеры для этой оси будут умножены на коэффициент масштабирования. Т.е. при любом коэффициенте масштабирования, отличающемся от единицы, параметры контура изменятся: в большую сторону при значении коэффициента > 1, в меньшую сторону при значении коэффициента < 1. Если значение коэффициента указано со знаком минус, то к масштабированию добавляется зеркальное отображение.

**Пример:**

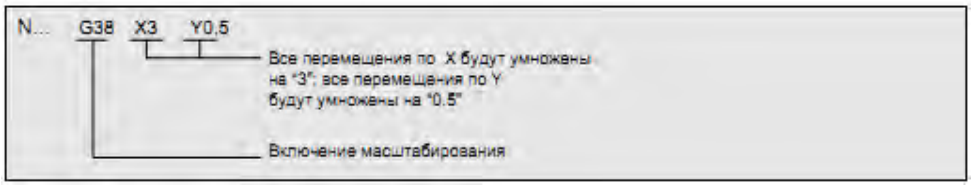




Рис.2.32.

Инструкция G39 выключает зеркальное отображение, масштабирование и поворот. Примеры масштабирования представлены на рис.2.33.

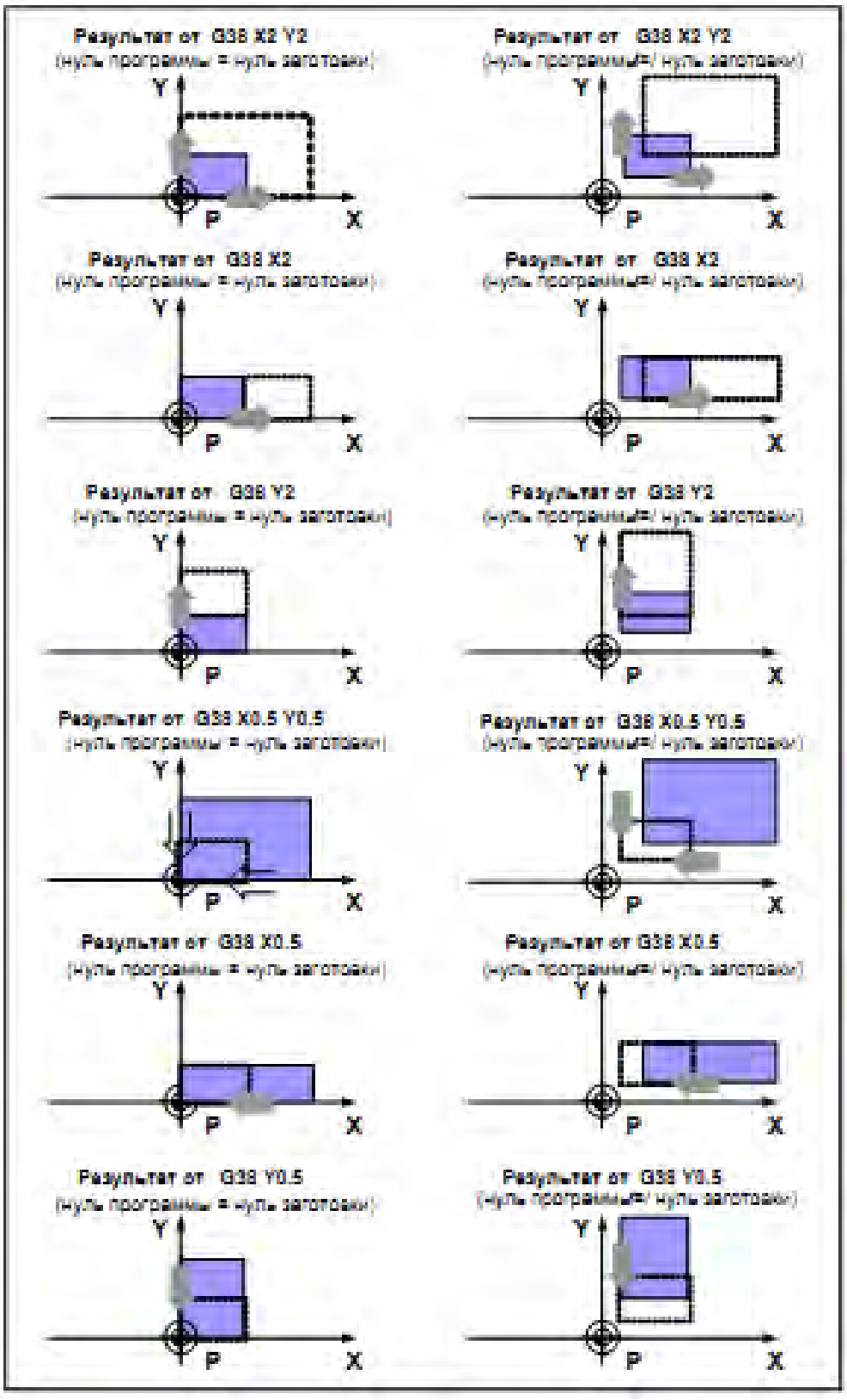
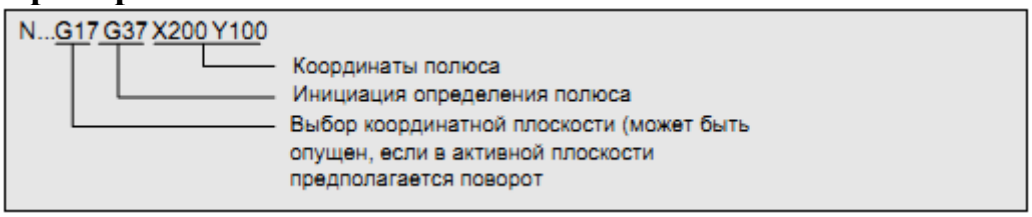

Результат от G38 X2 Y2
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 X2 Y2
(нуль программы= нуль заготовки)
Y
P
X
Результат от G38 X2
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 X2
(нуль программы= нуль заготовки)
Y
P
X
Результат от G38 Y2
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 Y2
(нуль программы= нуль заготовки)
Y
P
X
Результат от G38 X0.5 Y0.5
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 X0.5 Y0.5
(нуль программы= нуль заготовки)
Y
P
X
Результат от G38 X0.5
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 X0.5
(нуль программы= нуль заготовки)
Y
P
X
Результат от G38 Y0.5
(нуль программы = нуль заготовки)
Y
P
X
Результат от G38 Y0.5
(нуль программы= нуль заготовки)
Y
P
X

Рис.2. 33.

**24.3. Поворот,** - G37, G38, G39. Поворот осуществляется в активной плоскости соответственно инструкциям G17, G18, G19, G20.

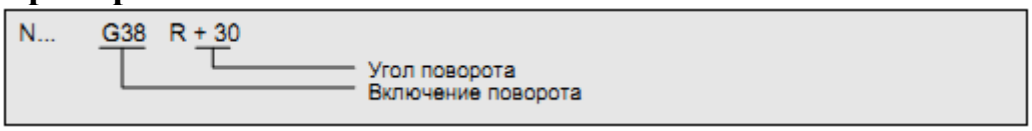
Модальная инструкция G37 служит для задания полюса поворота в абсолютных координатах относительно нуля управляющей программы. Если поворот осуществляется относительно этого нуля, то инструкция G37 не требуется. Действие инструкции отменяется инструкциями G39 или G37 (с другими координатами полюса).

**Пример:**




Модальная инструкция G38 активизирует поворот; при этом должен быть запрограммирован угол поворота радиуса. Положительные значения угла поворота радиуса указывают на вращение против часовой стрелки; отрицательные значения угла поворота радиуса указывают на вращение по часовой стрелке. Программное смещение G60 будет учтено при повороте для расчета координат.

**Пример:**




Пример поворота проиллюстрирован на рис.2.34.

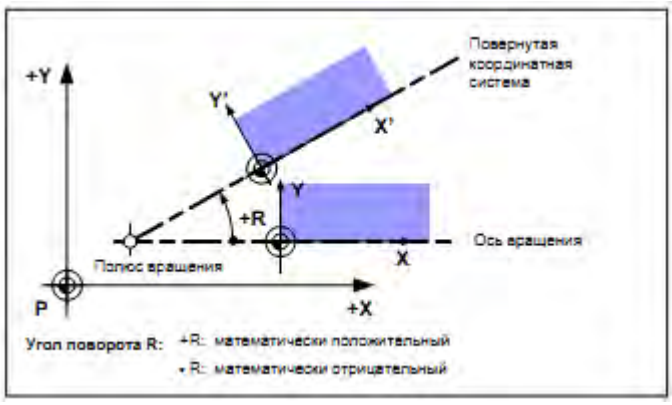



Рис.2.34.

Модальная инструкция G39 деактивирует зеркальное отображение, масштабирование и поворот.

Примеры использования инструкций поворота показаны на рис.2.35.

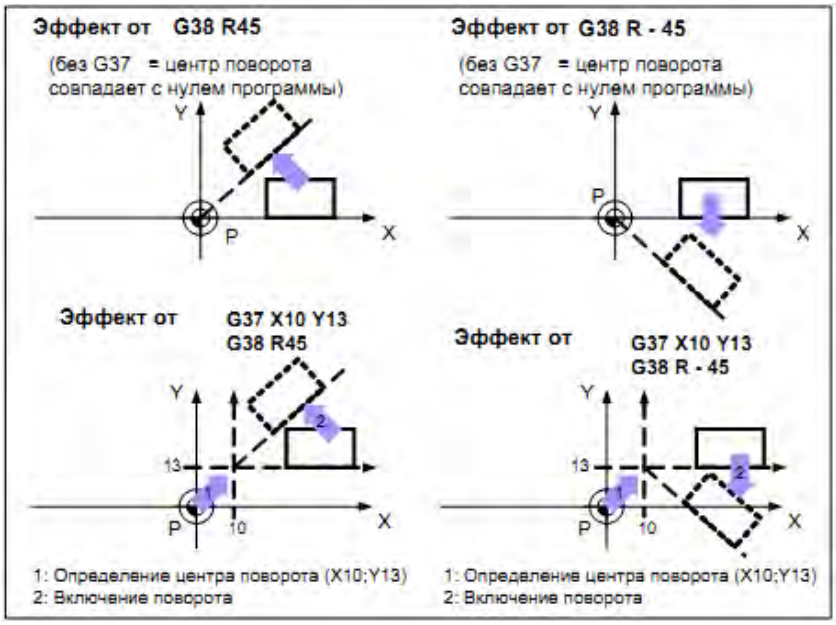



Рис.2. 35.

24.4. **Совместное использование зеркального отображения, масштабирования и поворота.** При совместном использовании, первым выполняется поворот, а затем зеркальное отображение и масштабирование.

**Пример**:

| | |
|---|---|
| N…G37 X100 Y-200 | /Определение полюса вращения и зеркального |
| | /отображения |
| N…G38 X-3 Y-2 R115 | /Угол поворота против часовой стрелки на 115 градусов. |
| | /Зеркальное отображение задано знаками минус. |
| | /Коэффициент масштабирования по осям X и Y |
| | /равен соответственно трем и двум. |
| N…G39 | /Все три функции деактивируются. |

**24.5. Отношения между инструкциями G37/G38, с одной стороны, и инструкциями G60 или G54, G259, - с другой стороны.** В пределах координатной системы управляющей программы инструкция G60 оказывает влияние на инструкции G37/G38, см. рис.2.36.

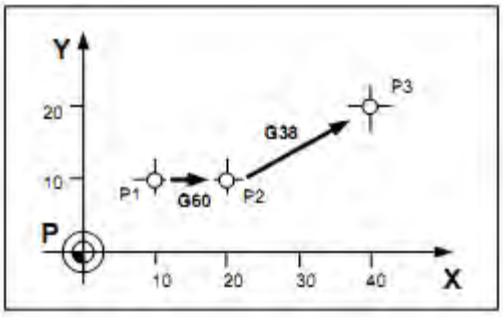



Рис.2.36.

Рисунок можно проиллюстрировать фрагментом программы.

```
N05…                    /Точка P1: исходное положение.
N10 G60                 /Точка P2: смещение G60 точки P1.
N20 G38 X2 Y2           /Активизация масштабирования.
N30 G01 X10 Y10         /Точка P3: масштабное отображение положения P2.
```

Инструкции G54, G259 инициируют смещение начала координатной системы управляющей программы по отношению к началу координатной системы станка. По этой причине эти инструкции не оказывают влияния на операции, предусмотренные инструкциями G37/G38 или G60, см. рис.2.37.

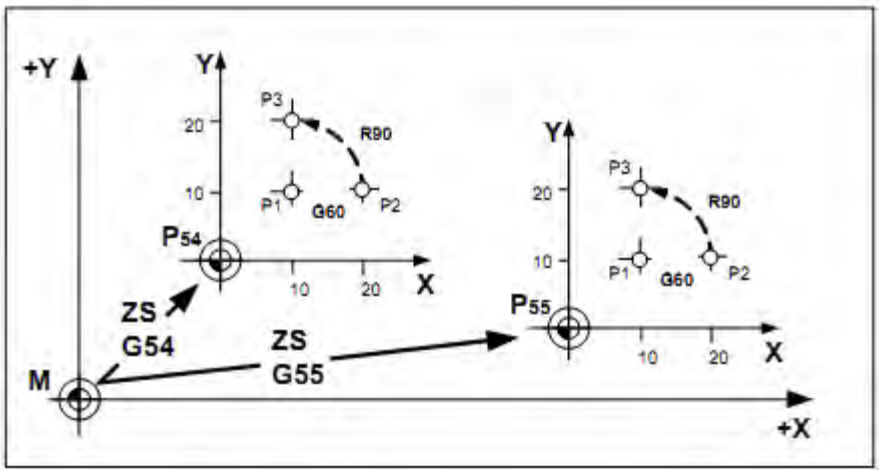



Рис.2. 37

| Пример для G54 | Пример для G55 | Комментарий |
|---|---|---|
| N10 G54 | N110 G55 | Вызов функции |
| N20 G37 X10 Y10 | N120 G37 X10 Y10 | Точка P1 есть полюс с координатами X10 Y10 |
| N30 G60 X10 | N130 G60 X10 | Точка P2 есть смещение G60 точки P1 |
| N40 G38 R90 | N140 G38 R90 | Точка P3 есть результат поворота P2 |
| N50 G01 X10 Y10 | N150 G01 X10 Y10 | |

**25. Эквидистантная коррекция,** - G40, G41, G42. В результате коррекции инструмент перемещается по траектории, параллельной исходному контуру. Принцип эквидистантной коррекции проиллюстрирован на рис.2.38.

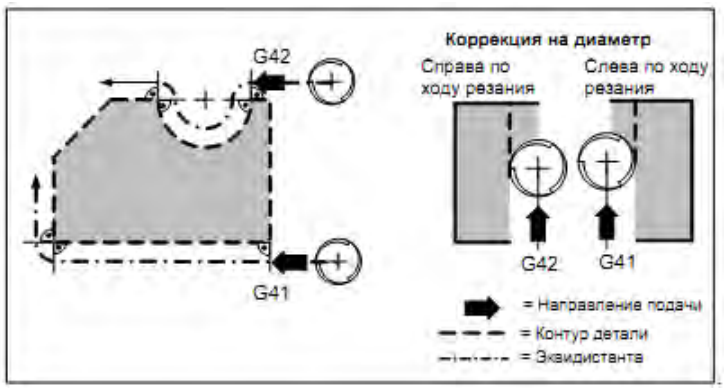



Рис.2.38.

Вдоль контура и тех сопряжений кадров, для которых угол наклона касательной остается неизменным, эквидистанта однозначно определяется параметрами контура, см. рис.39.

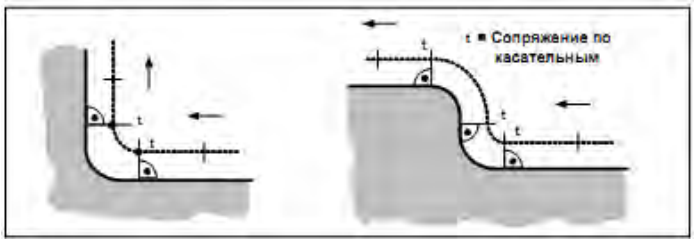



Рис.2.39.

В других же нерегулярных случаях внешних сопряжений кадров система ЧПУ рассчитывает сопряжения отрезков эквидистант соответственно инструкциям G68 или G69, см. рис.2.40.

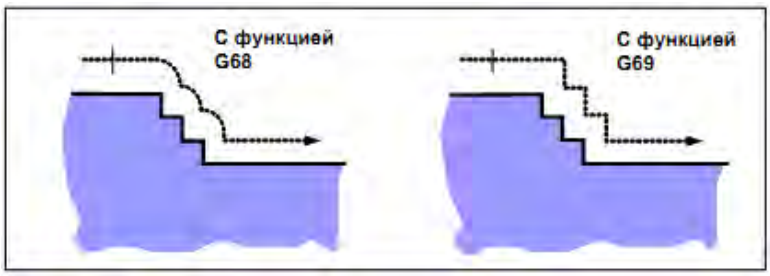



Рис. 2.40.

В случае нерегулярных сопряжений внутренних контуров система ЧПУ

рассчитывает пересечения эквидистант для определения нужной траектории, см. рис.2.41. В некоторых случаях это может привести к полному искажению контура. Чтобы избежать этого, некоторые системы ЧПУ располагают функцией «контроля коллизий

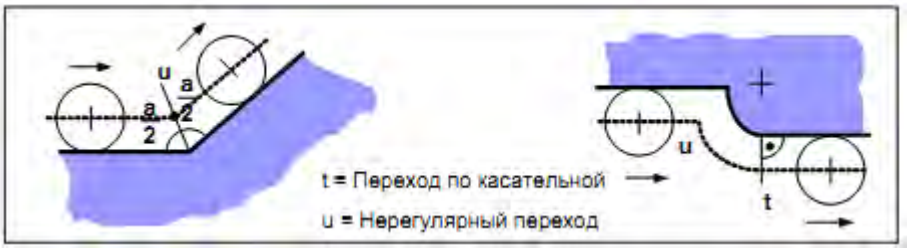



Рис. 2.41.

**25.1. Отмена коррекции,** - G40. Отмена коррекции G40 может сопровождаться прямолинейным движением в активной плоскости. В этом случае выход из эквидистантной траектории осуществляется «по пути» к конечной точке кадра. Если активны функции круговой интерполяции, то действие инструкции G40 не должно сопровождаться перемещением.

**25.2. Эквидистантная коррекция,** - G41. Инструкция G41 инициирует положительную эквидистантную коррекцию слева от заготовки, если смотреть в направлении подачи. Для реализации коррекции радиус фрезы программируют в D-слове, а номер инструмента в T-слове. Вместе с инструкцией G41 можно программировать линейные перемещения; тогда активизация эквидистантной коррекции произойдет «по пути» движения к конечной точке кадра.

**Пример:**

```
N60 G41 X…Y…Z…D…
или
N60 T123 M06
N65 G41 X…Y…Z…
```

**25.3. Эквидистантная коррекция,** - G42. Инструкция G42 инициирует эквидистантную коррекцию справа от заготовки, если смотреть в направлении подачи. Все остальное – идентично инструкции G41.

**Пример:**

```
N60 G42 X…Y…Z…D…
или
N60 T123 M06
N65 G42 X…Y…Z…
```

**26. Смещение нуля (ZS), - отмена смещения G53; инициация смещения G54-G59; отмена первого аддитивного смещения G153; инициация первого аддитивного смещения G154-G159; отмена второго аддитивного смещения G253; инициация второго аддитивного смещения** G254-G259. Инструкция смещения нуля позволяет сместить начало координат управляющей программы по отношению к началу координат станка. Значения смещений сохраняются в таблицах. Каждая таблица может содержать до трех групп из шести смещений нуля соответственно инструкциям G54, G59, G154, G159, G254, G259. Для активизации смещения нуля необходимо выбрать желаемую таблицу (см. G22), а далее просто упомянуть соответствующую G-инструкцию, без какой либо дополнительной позиционной информации. Все смещения нуля действуют аддитивно: G54+G156+G259. Смещения нуля внутри группы обновляют друг друга.

**Пример**

```
N…G22 V1          /Инициация таблицы V1 смещений нуля.
N… G54            /Смещение нуля активизировано, без функции перемещения.
или
N…G54 X…Y…Z…     /Смещение нуля связано с приведенной здесь позиционной
                  /информацией.
```

```
N…
N…G154 X…Y… Z…   /Первое аддитивное смещение нуля связано с приведенной
                  /здесь позиционной информацией.
N…
N…G254 X…Y…Z…    /Второе аддитивное смещение нуля связано с приведенной
                  /здесь позиционной информацией.
N…
N…G253           /Отменяется второе аддитивное смещение нуля.
N…
N…G53            /Отменяются все активные смещения нуля.
```

Инструкции G54 до G59 являются модальными и взаимно деактивируют друг от друга.

Инструкция G53 отменяет смещения нуля, объявленные инструкциями этой группы и инструкциями групп первого и второго аддитивного смещения. Инструкция G53 не оказывает влияния на программное смещение контура, заданное инструкцией G60.

Инструкции от G154 до G159 являются модальными и взаимно деактивируют друг друга. Их действие прекращается инструкциями G153 или G53. Инструкции от G254 до G259 являются модальными и взаимно деактивируют друг друга. Их действие прекращается инструкциями G253 или G53.

Принцип программирования смещения нуля проиллюстрирован на рис.2. 42.

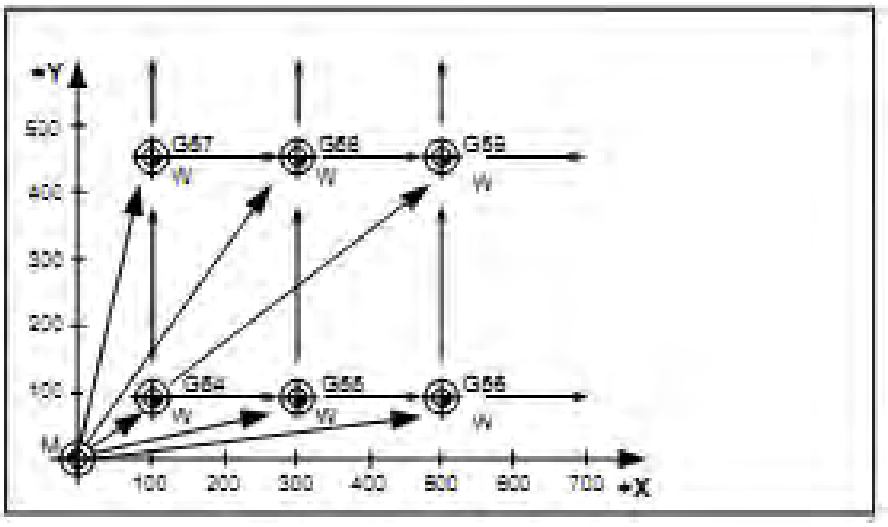



Рис.2.42.

**27. Программное смещение контура,** - G60, G67. Инструкция G60 не меняет положения координатной системы управляющей программы относительно координатной системы станка, но лишь осуществляет смещение контура в пределах координатной системы управляющей программы. Инструкция G60 не инициирует никаких перемещений.

Инструкция G60 включает программное смещение контура, а инструкция G67 – выключает это смещение.

**Пример:**

```
N10 G60 X10 Y10 Z50   /Новая нулевая точка программы имеет координаты X10,
                      /Y10, Z50. Перемещения в кадре отсутствуют.
N…
N100 G01 X…Y…Z…       /Перемещения с учетом программного смещения нуля.
N…
N210 G67 X…Y…Z…       /Перемещения без учета программного смещения нуля.
N…
Или:
N210 G67              /Сброс G60
```

При использовании инструкции G60, те координаты, которые не перепрограммируются, сохраняют прежние значения программного смещения.

**Пример:**

```
N10 G60 X10 Y10 Z50  /Новый старт программы в точке с координатами X10,
                     /Y10, Z50.  Перемещения в кадре отсутствуют.
N…
N100 G01 X…Y…Z…  /Перемещения с учетом программного смещения.
N…
N110 G60 X20 Y20     / Новый старт программы в точке с координатами X20,
                     /Y20, Z50. Смещение по  Z остается таким же, как и при
                     /предыдущем использовании инструкции G60!
                     /Перемещения в кадре отсутствуют.
N120 G01 X…Y… Z…/Перемещение с учетом программного смещения.
N…
N210 G67             /Сброс G60.
```

**28. Точное позиционирование,** - G61, G62, G163. Влияние динамики исполнительных органов станка таково, что образуется временное рассогласование между запрограммированными и фактическими координатами. Величина рассогласования для каждой оси зависит от скорости подачи и коэффициента «KV» усиления следящего привода по скорости. При неплавном переходе на стыке кадров (в уголках) рассогласование приводит к искажению контура.

Инструкция G61 позволяет избежать этого. Три опции точного позиционирования могут быть заданы инструкциями от G164 до G166.

Инструкция G61 работает только при движении со скоростью подачи. Точное позиционирование при ускоренном перемещении осуществляют с помощью инструкций.

Инструкция G61 работает только при движении со скоростью подачи. Точное позиционирование при ускоренном перемещении осуществляют с помощью инструкций G161/G162. Влияние инструкции G163 является превалирующим: она заменяет G161/G162 и работает при ускоренном движении и движении соскоростью подачи; инструкция G62 выключает точное позиционирование (см. рис.2.43); инструкция G163 включает точное позиционирование при ускоренном движении и движении со скоростью подачи. Инструкции G61, G62 и G163 являются модальными.

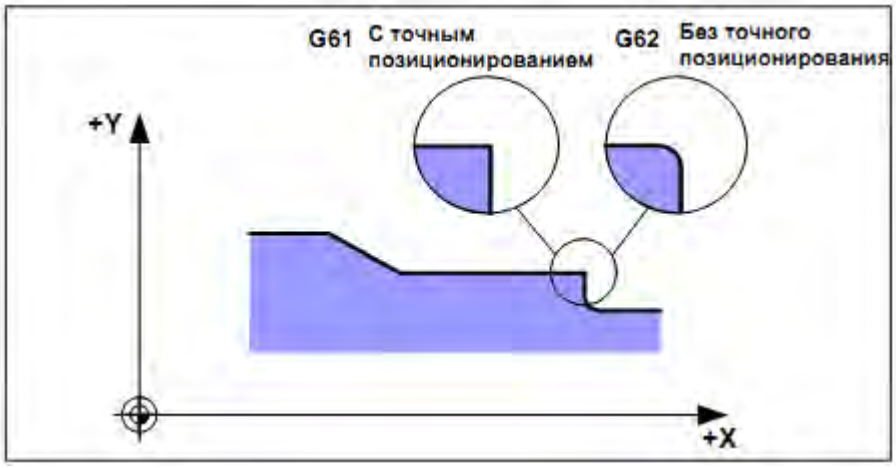



Рис. 2.43.

**Пример**

```
N10  G61          /Включение точного позиционирования. Перемещений нет.
N11  G01 Y200     /Линейная интерполяция с точным позиционированием.
Или:
N10  G62          /Выключение точного позиционирования.
N11  G01 Y200     /Линейная интерполяция без точного позиционирования.
N50  G61 X200     /Линейная интерполяция с точным позиционированием уже
                  /в этом кадре.
```

**29. Работа с потенциометром «проценты от скорости»,** - G63, G66. Инструкции используют для программного влияния на потенциометры процентов от скорости подачи и ускоренного перемещения, потенциометр частоты вращения шпинделя. Обе инструкции работают в автоматическом режиме и режиме ручного управления. Инструкция G63 включает 100% от запрограммированного значения скорости, независимо от фактического положения потенциометра, т.е. потенциометр деактивируется. Инструкция G66 активизирует значение скорости, заданное потенциометром. Обе инструкции модальны и взаимно исключают одна другую.

**Пример включения 100% от запрограммированной скорости:**

```
N ...G63 G01 X120.675 Y34.896 Z-34.765 F200 S1000 M04
```

**30. Привязывание скорости подачи, - к точке контакта фрезы и детали,** - G64; **к центру фрезы,** - G65. Скорость подачи поддерживается постоянной либо в точке контакта инструмента, либо в центре фрезы, см. рис.2.44.

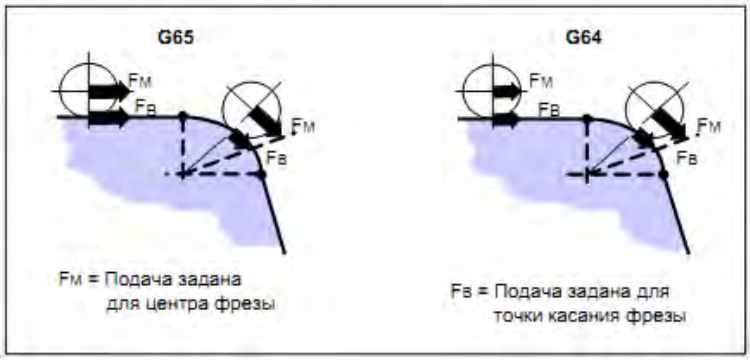



Рис.2.44.

Применять инструкцию G64 можно только в том случае, если активны инструкции эквидистантной коррекции G41/G42 и используется круговая интерполяция G2/03/05. Инструкции G64 и G65 модальны и взаимно исключают одна другую.

В зоне так называемых «компенсирующих дуг» (например при обходе угла, см. рис.2.45) фактическое значение скорости подачи зависит от того, в каком кадре F-слово запрограммировано, см. примеры.

**Пример 1:**

```
N10 G64 X100 F100
N20 Y100 F200          /Подача в зоне компенсирующей дуги равна F100.
```

**Пример 2:**

```
N10 G64X100  F100
N20 F200
N30 Y100               /Подача в зоне компенсирующей дуги равна F200.
```

**Пример 3:**

```
N10 G64 X100 F100
N20 Z50
N30 Y100 F200          /Подача в зоне компенсирующей дуги равна F100.
```

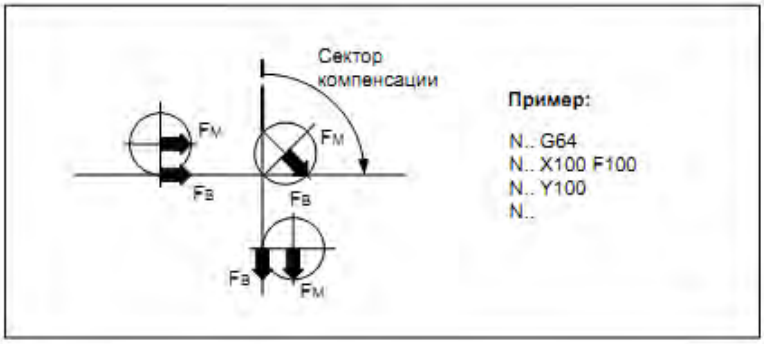



Рис.2.45.

**31. Сопряжение эквидистант на стыке кадров, - по дуге, - G68; по траектории пересечения эквидистант,** - G69. Инструкции являются модальными и работают при активной эквидистантной коррекции. Их действие сводится к автоматической генерации дуги (G68) или траектории пересечения эквидистант на стыке «не плавно» сопрягаемых кадров.

Инструкция G68 инициирует автоматическое соединение разрыва эквидистант с помощью дуги радиуса r , см рис.2.46.


Рис.2.46.

Инструкция G69 инициирует автоматическое соединение разрыва эквидистант по траектории пересечения эквидистант (см. рис.47). И здесь возможны две ситуации. В первой из них пересечение существует. В зависимости от расстояния **A** между точкой **КЕ** пересечения исходного контура и точкой S пересечения эквидистант, соединение разрыва осуществляется различными способами (см. рис.2.47).

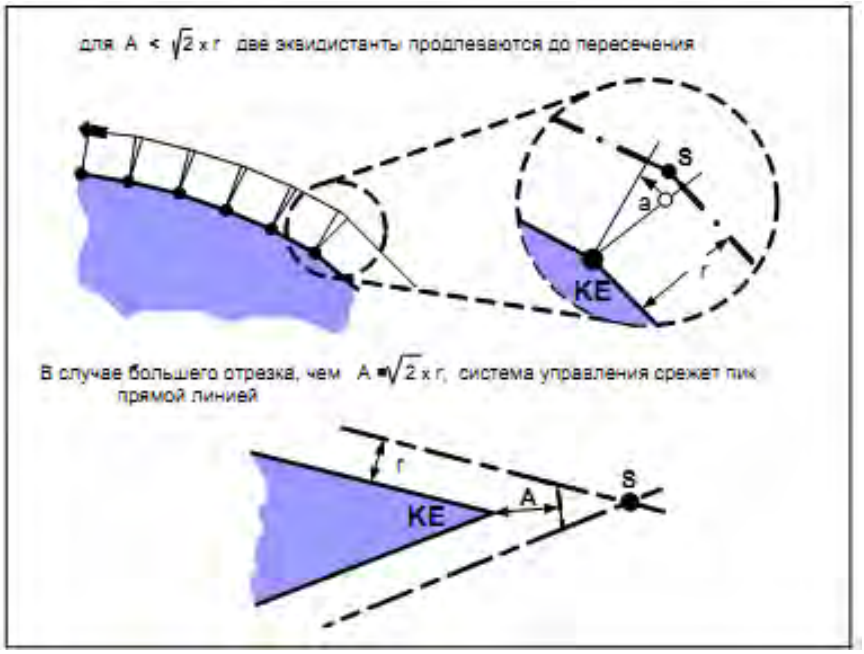



Рис.2.47.

Во второй ситуации эквидистанты не пересекаются. Это бывает, если сопряжение кадров с прямолинейным и круговым движением имеет разрыв в производных от исходных контуров. Это бывает также при «не гладком сопряжении двух дуг», см. рис.48. При этом, инструкция G68 инициирует автоматическое сопряжение контура по дуге радиуса r, см. рис. 2.48.

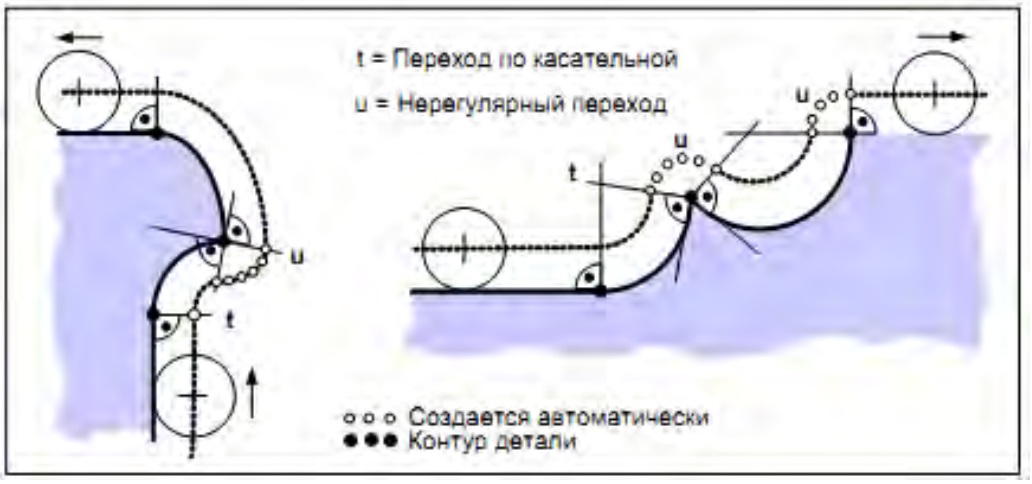



Рис. 2.48.

**32. Программирование в дюймах,** - G70. Инструкция G70, означает, что программирование перемещений и подачи использует дюймовую систему измерения. Все активные метрические данные и смещения нуля автоматически конвертируются в дюймы. Инструкция G70 является модальной, а ее действие прекращается инструкцией G71.

**33. Метрическое программирование,** - G71. Инструкция G71, означает, что программирование перемещений и подачи использует метрическую систему измерения. Все активные дюймовые данные и смещения нуля автоматически конвертируются в метрические данные и смещения. Инструкция G71 является модальной, а ее действие прекращается инструкцией G70.

**34. Линейная интерполяция с точным позиционированием,** - G73. В отличие от кадра с инструкцией G01, кадр с инструкцией G73 всегда выполняется с точным позиционированием, независимо от инструкций G61/G62. Глобальное определение опции точного позиционирования устанавливается с помощью инструкций от G164 до G166. Инструкция G73 прекращает действие инструкций G0O, G01, G02, G03, G05, G10-G13 и G200.

**35. Выход в начало координат,** - G74. Инструкция G74 инициирует одновременный выход в начало координат для тех координатных осей, которые указаны в том же кадре. Инструкция G74 действует только в том

кадре, в котором упомянута. При этом все активные позиционные данные и программные смещения сохраняются.

Адреса осей, по которым осуществляется выход в начало координат, сопровождаются нулевыми значениями "XO", "YO", "ZO".

**Пример:**

```
N100 G74 XO YO ZO   / Выход в начало координат осуществляется одновременно
                    / по  X, Y и Z.
```

**36. Работа с датчиком касания,** - G75. Измерительная головка триггерного типа перемещается вдоль выбранной оси со скоростью подачи на величину запрограммированного перемещения в направлении детали. После остановки проверяется факт касания с деталью; и если касания нет, то вызывается измерительный цикл, который организует перемещение до касания. После касания (т.е. после срабатывания триггерной головки), читаются показания датчика положения следящего привода, и пройденный путь с учетом коррекции по результатам измерительного цикла сохраняется в памяти. Инструкция G75 действует подобным образом только в текущем кадре.

**Пример:**

```
N100 G75 X400
```

**37. Переключение кадров высокоскоростным внешним сигналом,** (HS, High Speed), - G575. Инструкция G575 позволяет перейти к следующему кадру управляющей программы до завершения текущего кадра при возникновении HS-сигнала на входном регистре системы ЧПУ. Перемещение в текущем кадре программируют так, что оно заканчивается за пределами контура; таким образом, кадр никогда не может быть выполнен до конца. Необходимое перемещение непрерывно измеряется, и в нужный момент формируется HS-сигнал, инициирующий переход к очередному кадру. Инструкция работает с кадрами, в которых предусмотрено линейное перемещение (G0O, G01, G10, G11, G73, G200). Инструкция G575 работает в двух вариантах: без изменения координат точки завершения перемещения, несмотря на прерывание текущего кадра; а также и с игнорированием остатка неотработанного кадра.

**37.1. Сохранение координат точки завершения перемещения,** - G575 HS<x>=<y>. В определении формата инструкции <x> означает номер одного из восьми возможных высокоскоростных сигналов (HS1-HS8); а <y> принимает значения нулевого (0 В) или высокого уровня (24 В) напряжения на входном регистре.

**Пример (см. рис.**2.49):

```
N20 G00 X0 Y0
N30 G575 G01 X70 Y7 F1000 HS1=1   /Перемещение по оси X осуществляется
                                  /со скоростью подачи F1000 до тех пор,
                                  /пока уровень HS-сигнала номера 1 не
                                  /станет высоким или само
                                  /перемещение не выйдет в координаты
                                  /X70 Y7.
N40 G575 X90 Y9 F900 HS1 =0       /Перемещение по оси X осуществляется
                                  /со скоростью подачи F 900 до тех пор,
                                  /пока уровень HS-сигнала номера 1 не
                                  /станет низким или само
                                  /перемещение не выйдет в координаты
                                  /X90 Y9.
N50 X100 Y10 F800                 /Перемещение до окончательной точки
                                  /X100 Y10 выполняется со скоростью
                                  /подачи F800.
```

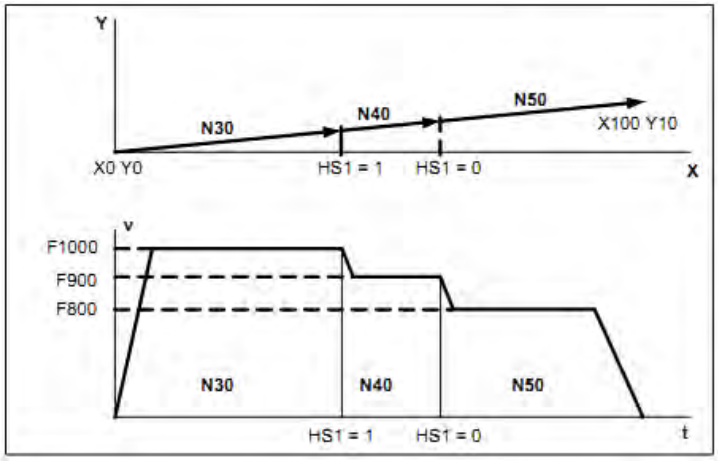



Рис. 2.49.

При использовании варианта инструкции G575 HS<x>=<y> может произойти искажение траектории, но точка выхода из траектории останется неизменной.

**Пример (см. рис.**2.50):

```
N05 G01   F100 X0 Y5              /Координаты конечной точки X0 Y5.
N10 G575 HS1=1 G90 X50            /Координаты конечной точки X50 Y5.
N20 G575 HS1=0 G91 Y10            /Координаты конечной точки X50 Y15
                                  /(Абсол. величина   Y5 + прираш. Y10).
N30 G91 Y5                        /Координаты конечной точки X0 Y5
                                  /(Абсол. величина   Y15 + прираш. Y5).
```

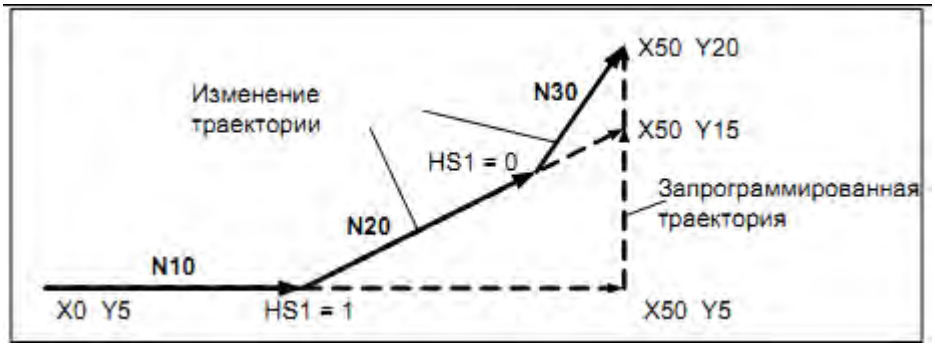



Рис. 2.50.

**37.2. Игнорирование остатка неотработанного кадра,** - G575 HS<x>=<y> HSSTOP=<z>. При достижении сигналом HS<x> значения <y> скорость текущего кадра снижается до нуля (HSSTOP=0) или уменьшается скачкообразно (HSSTOP= -1); при этом не пройденный путь игнорируется (пропадает). Таким образом, конечная точка текущего кадра и начальная точка следующего кадра определяются событием, обозначенным HS-сигналом. Имя HS-сигнала <x> может изменяться от 0до 8. Уровень HS-сигнала <y> принимает значения 0 или 1. Параметр торможения равен 0 или –1.

**Пример (см.рис.**2.51):

```
N05 G01 F1000 X0 Y5                  /Координаты конечной точки X0 Y5.
N10 G575 HS1=1 HSSTOP=0 G90 X50      /Внешнее событие при X меньше 50.
N20 G575 HS1=0 HSSTOP1=0 G91 Y10     / Внешнее событие при Y меньше 10.
N30 G91 Y5                           /Координаты конечной точки: приращение
                                     /на Y5 вслед за последним событием, Y
                                     /меньше 10.
```

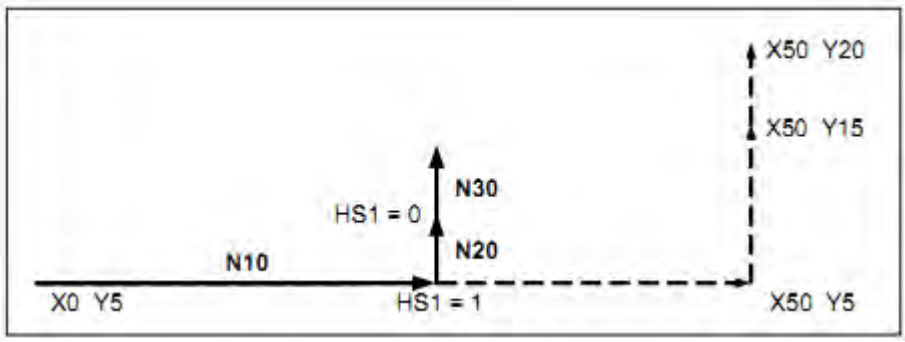



Рис.2.51.

**38. Перемещение в точку с абсолютными координатами в системе координат станка,** - G76. Инструкция G76 организует подобное перемещение, например, для смены или контроля целостности инструмента, для запуска измерительных циклов, для смены палет. Эта инструкция

действует только в одном кадре. Перед началом перемещения деактивируются различные компенсации, смещения нуля, функции зеркального отображения и др. Все деактивированные функции восстанавливаются в очередном кадре.

**39. Управление сверлильными осями,** - G78, G79. Независимо от кинематики станка, любая его ось может быть сверлильной. Инструкция G78 активизирует сверлильную ось и соответствующие коррекции инструмента. Для организации коррекции предусмотрены две «компенсационные группы». Первая группа содержит описание коррекции (H) из внутренних таблиц системы ЧПУ и описание внешней (external) коррекции ($H_{ext}$) со стороны программируемого контроллера (оба вида коррекции суммируются). При этом внешняя коррекция связана с использованием инструкций G145-G845. Коррекции L1, L2, L3 второй компенсационной группы соотнесены к осям; они связаны с использованием инструкций G147-G847.

Синтаксис слова с инструкцией G78 выглядит следующим образом:

G78 <имя оси i> <коррекция i>...[<имя оси n> <коррекция n>]

**Здесь: имена осей i…n являются логическими адресами координатных осей; значения <коррекции i > указаны в таблице. Знак + или – в формате коррекции указывает на направление компенсации размера инструмента.**

| № | Формат <коррекции i> | Комментарий |
|---|---|---|
| 1. | +/-1 или +/-13 | Первая компенсационная группа H или Hext оси i |
| 2. | +/-21 | Вторая компенсационная группа, первая коррекция длины L1ext оси i |
| 3. | +/-22 | Вторая компенсационная группа, вторая коррекция длины L2ext оси i |
| 4. | +/-23 | Вторая компенсационная группа, третья коррекция длины L3ext оси i |

Коррекции для нескольких осей могут быть включены в кадре одной инструкцией G78. Инструкция G79 деактивирует одну сверлильную ось или все сразу соответственно формату G79 {<CG> i}; где <CG> i указывает на номера компенсационных групп и коррекции, а также и ось i (CG – Compensation Group).

**Пример:**

| | |
|---|---|
| G78 X-1 | /Ось X объявлена сверлильной с отрицательной |
| | /коррекцией на длину инструмента. |
| G78Y1 | /Ось Y объявлена сверлильной с положительной коррекцией |
| | /на длину инструмента. |
| G78 YA21 YB22 | /Оси YA и YB объявлены сверлильными; предусмотрена |
| | /вторая компенсационная группа с положительными |
| | /коррекциями L1 и L2. |
| G79 | /Сверлильные оси деактивированы. |

**40. Стандартные сверлильные циклы,** - G80-G86, G184. Все стандартные циклы запрограммированы заранее, их вызывают соответствующими инструкциями с указанием необходимых параметров. Обобщенная последовательность движений в стандартном цикле представлена на рис.2.52.

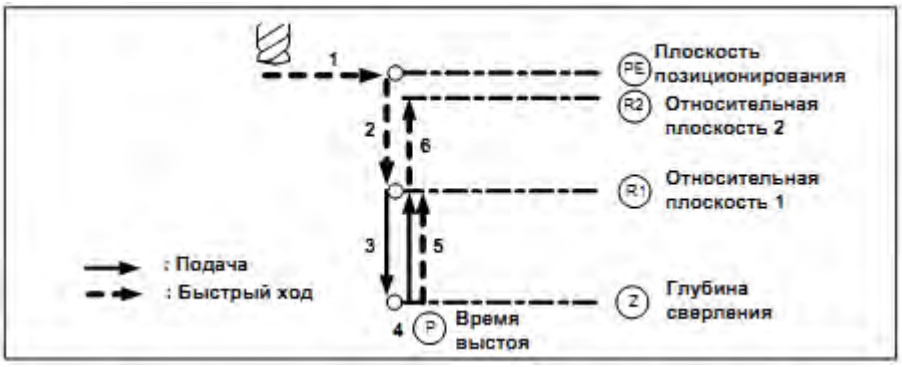



Рис.2.52.

В схеме на рис.2.52 предусмотрены следующие движения:

1. Ускоренное позиционирование в активной плоскости.
2. Ускоренный подвод к точке R1 в «безопасной плоскости».
3. Рабочее движение вдоль оси Z на глубину сверления со скоростью подачи.
4. Выстой в течение времени P для торможения шпинделя перед его реверсом.
5. Вывод инструмента со скоростью подачи или ускоренно к точке R1 в «безопасной плоскости».
6. Возможный ускоренный отвод к точке R2.

Параметры стандартного цикла должны быть специфицированы вслед за инструкцией стандартного цикла. Число параметров зависит от цикла, причем порядок их объявления строго определен. Все параметры должны быть заданы внутри квадратных скобок «[« и «]» и разделены запятыми.
Обзор используемых форматов стандартных циклов (некоторые параметры показаны на рис.2.53):

```
G80: N…G80                                  /Выключение активного стандартного цикла
G81: N…X…Y…G81 [Z, R1, P, R2]               /Включение цикла G81
G82: N…X…Y…G82 [Z, R1, P, R2]               /Включение цикла G82
G83: N…X…Y…G83 [Z, R1, K, k, P, R2]          /Включение цикла G83
G84: N…X…Y…G84 [Z, R1, P, R2]               /Включение цикла G84
G85: N…X…Y…G85 [Z, R1, P, R2]               /Включение цикла G85
G86: N…X…Y…G86 [Z, R1, P, R2]               /Включение цикла G86
G184: N…X…Y…G184 [Z, R1, PR2, GS, U1, U2]   /Включение цикла G184
```

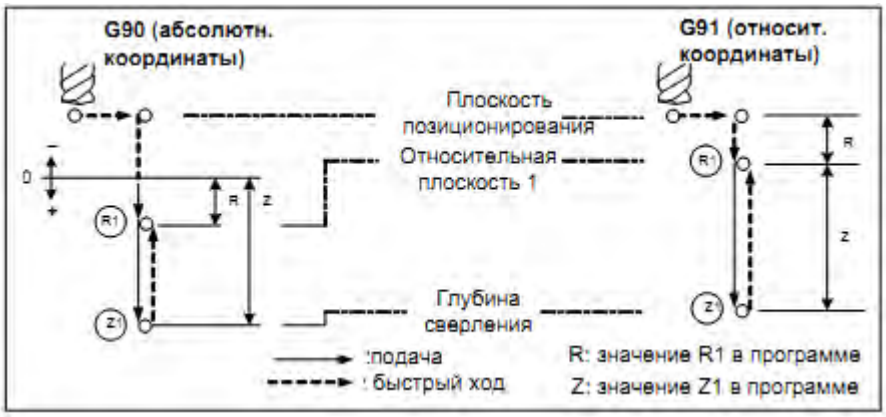



Рис.2.53.

**40.1. Цикл серления,** - G81. Содержание цикла – зацентровка и сверление. После достижения глубины врезания, осуществляется выстой. Выход производится на ускоренной подаче (см. рис.2.54).

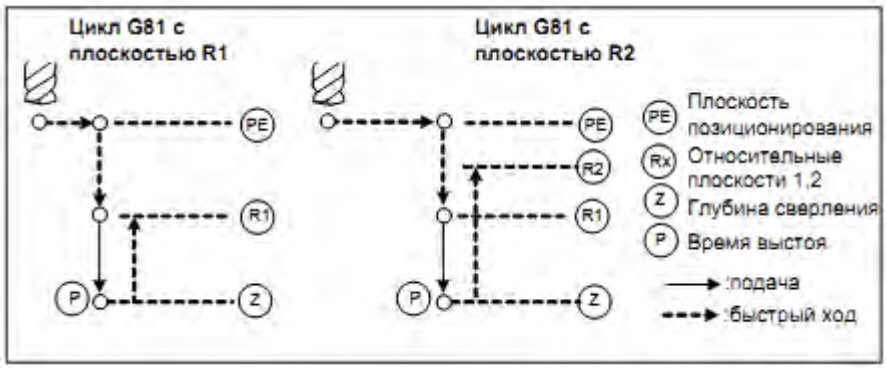



Рис.2.54.

```
N100 X…Y…G81 [Z, R1, P, R2]
```

**40.2. Цикл сверления,** - G82. Цикл аналогичен G81. Однако выход в точку R1 осуществляется со скоростью рабочей подачи (см. рис.2.55).

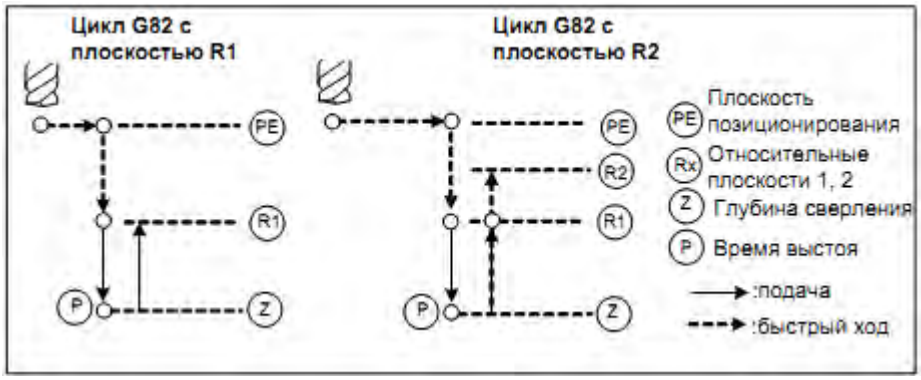



Рис. 2.55.

```
N100 X…Y…G82 [Z, R1, P, R2]
```

**40.3. Цикл глубокого сверления,** - G83. Цикл предполагает полное удаление стружки из отверстия. После каждого очередного врезания на глубину **K** осуществляется ускоренный вывод сверла в безопасную плоскость R1. Далее выполняется очередной ускоренный ввод сверла на глубину k, где ускоренная подача меняется на рабочую. Пошаговые углубления повторяются до достижения запрограммированной глубины Z (см. рис.2.56).

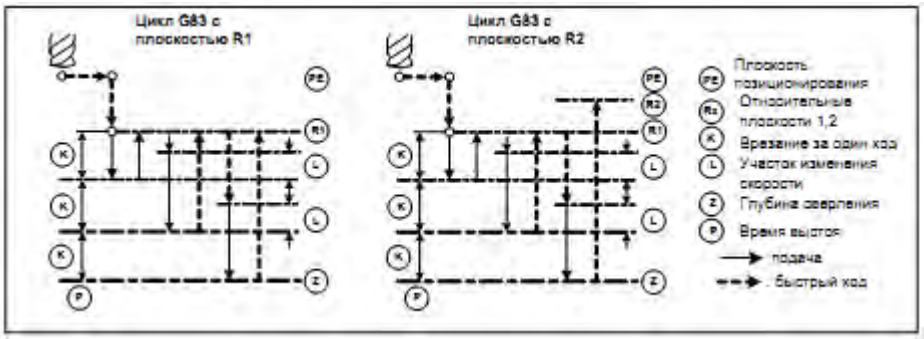



Рис.2.56.

```
N100…X…Y…G83 [Z, R1, K, k, P, R2]
```

**40.4. Нарезание резьбы с компенсирующим патроном,** - G84. Инструкция инициирует нарезание левой или правой резьбы. Врезание метчика происходит за счет вращения шпинделя по часовой стрелке (вспомогательная функция M3) или против часовой стрелки (вспомогательная функция M4). По достижении запрограммированной глубины Z, направление вращения шпинделя изменяется, при этом может быть предусмотрена выдержка времени P (см. рис.2.57).

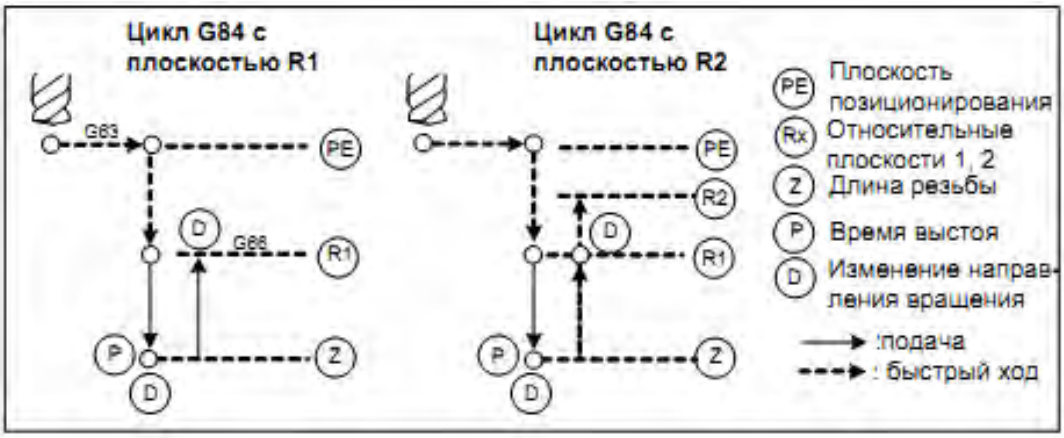



Рис.2.57.

```
N100…X…Y…G84 [Z, R1, P, R2]
```

**40.5. Нарезание резьбы без компенсирующего патрона,** - G184. Предусловием служит использование инструкции G32. Подача подсчитывается как произведение частоты вращения шпинделя на шаг резьбы. Левая или правая резьба выбирается с помощью знака параметра GS (шага резьбы). По достижении глубины резьбы Z, направление вращения шпинделя изменяется. Следовательно, вывод инструмента осуществляется со скоростью рабочей подачи (см. рис.2.58).

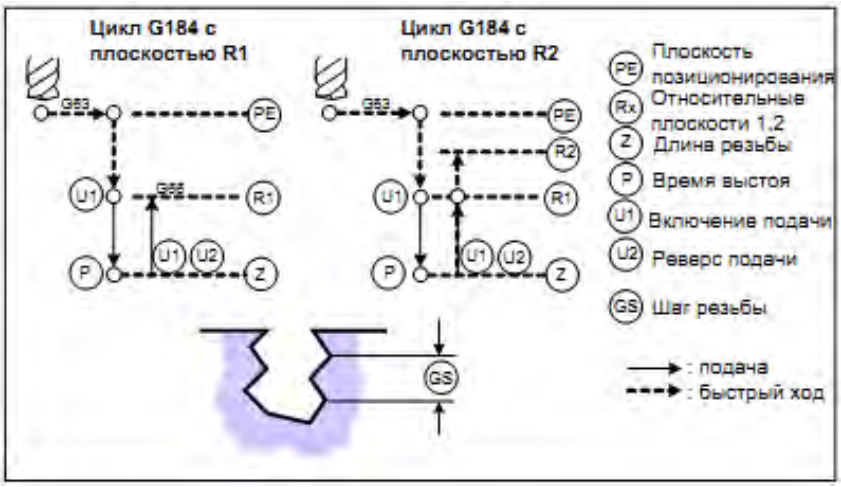



Рис.2. 58.

Примеры

```
N100…X…Y…G184 [Z, R1, P, R2, GS, U1, U2, RP]     /Правая резьба.
```

```
N100…X…Y…G184 [Z, R1, P, R2, -GS, U1, U2, RP]     /Левая резьба.
```

Параметр RP определяет угловую ориентацию шпинделя.

**40.6. Рассверливание,** - G85. По достижении заданной глубины Z, шпиндель останавливается. Далее возможна выдержка времени. Затем осуществляется ускоренный вывод инструмента (см. рис. 2.59).

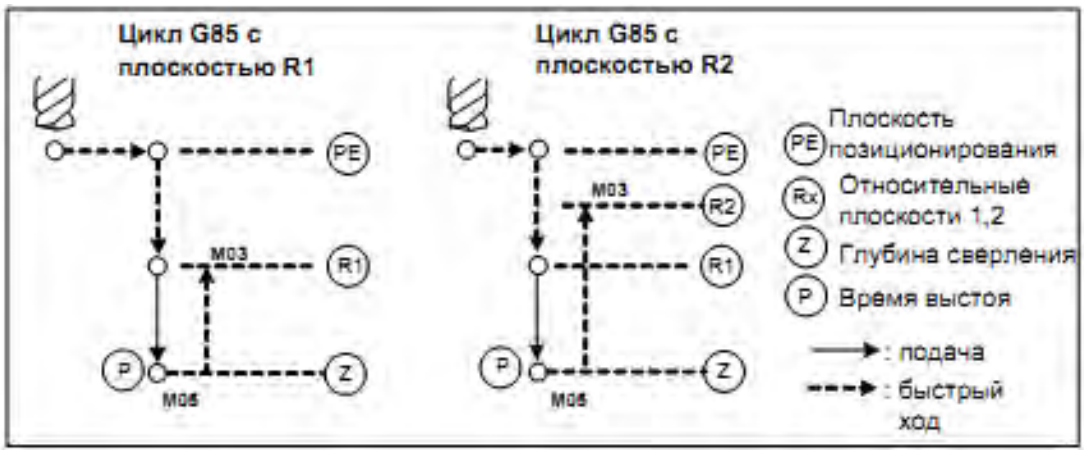



Рис.2.59.

N100…X…Y…G85 [Z, R1, P, R2]

**40.7. Рассверливание с выводом инструмента со скоростью рабочей подачи,** - G86. Отличается от предыдущего цикла лишь тем, что возврат к плоскости R1 осуществляется со скоростью рабочей подачи (см. рис.2.60).

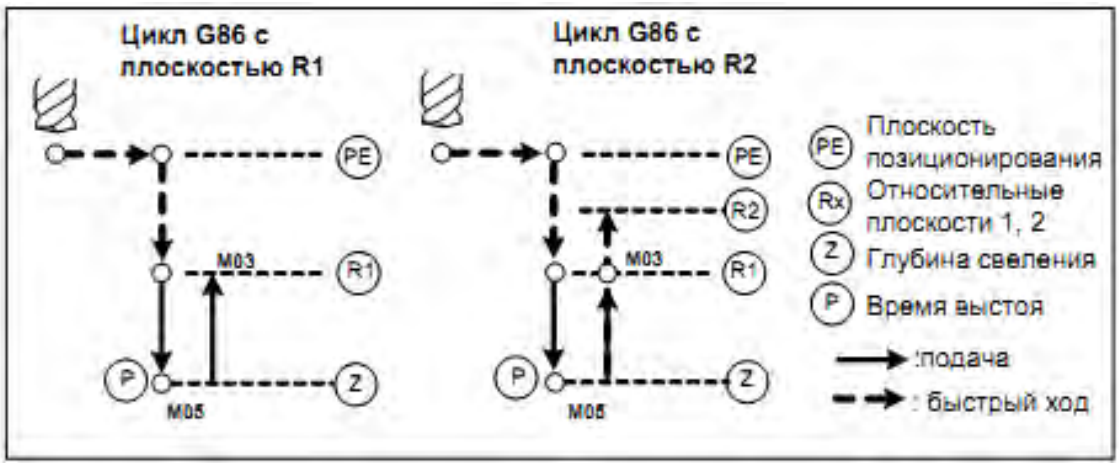



Рис.2.60.

N100…X…Y…G86 [Z, R1, P, R2]

### 40.8. Примеры программирования стандартных циклов.

**Пример 1.**

```
N90 G01 M3 S1050 F400
N91 G81 [-1000,-800]                /Вызов цикла без позиционирования.
N92 X600 Y800                       /Сверление начинается в этом кадре.

N95 X500 Y700 G81 [-1000,-800]      /Вызов цикла с позиционированием.
                                    /Сверление начинается в этом кадре.
N96 X600 Y800
N100 X800 Y700 G81 [-1000, -800.. -600]    /Возврат к плоскости
                                    /R2, выдержка времени отсутствует.

N110 X0 Y0 G81 [-1000,-800]         /Возврат к плоскости R1, выдержка времени
                                    /отсутствует.
N111 X-100  Y-500
N150 X-400  Y200 G81 [-1000,-800, 1]        /Возврат к плоскости R1, выдержка
                                    /времени 1 Сек.
N151 X200 Y300
```

**Пример 2: вызов стандартного цикла в главной программе, позиция выхода к точке начала цикла запрограммирована в подпрограмме (см. рис.**2.61).

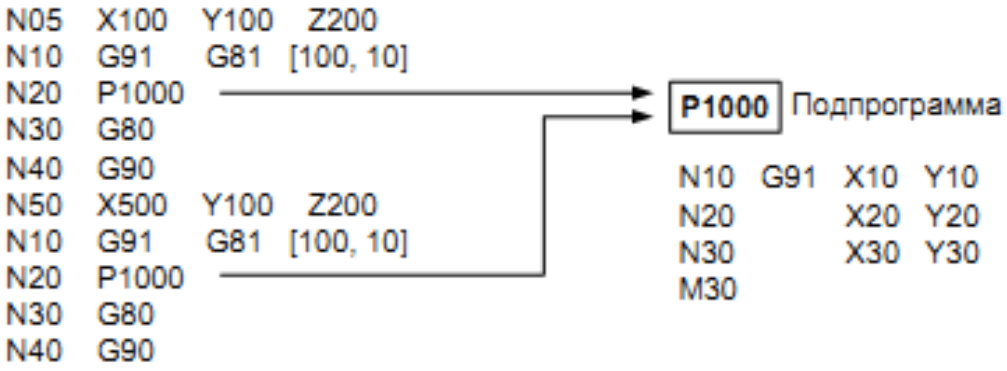



Рис.2.61.

**Пример 3: включение сверлильной оси X с положительной компенсацией длины инструмента.**

```
N10 G78 X1
N20 G01 M3 S1050 F400
N30 G81 [-100, -800]
N40 Y500 Z700
N50 G80
N60 G79
```

**41. Программирование в абсолютных координатах,** - G90. **Программирование в относительных координатах,** - G91. **Программирование в абсолютных координатах для «бесконечных осей»,** - G189. Инструкция G90 будет интерпретировать перемещения как абсолютные значения по отношению к активной нулевой точке. Инструкция G91 будет интерпретировать перемещения как приращения по отношению к ранее достигнутым положениям. Инструкция G189 будет интерпретировать перемещения как абсолютные по отношению к активной нулевой точке для бесконечных осей (например, осей вращения). Рис.2.62 демонстрирует различие инструкций G90 и G91.

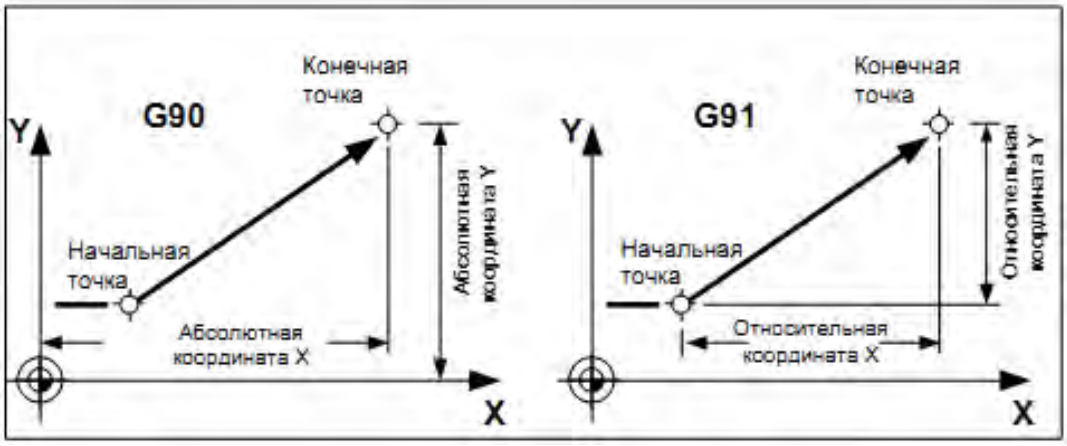



Рис.2. 62.

Инструкции G90, G91, G189 являются модальными и относятся к той же группе подготовительных функций, что и G190 G191.

**Пример:**

| | |
|---|---|
| N10 G90 | /Все размеры интерпретируются как абсолютные по /отношению к активной нулевой точке. |
| N20 X100 Y100 | /Текущие абсолютные координаты составляют / X100 Y100 |
| N30 G91 | /Все размеры интерпретируются как приращения к /ранее достигнутым координатам. |
| N40 X50Y10 | /Абсолютные текущие координаты равны X150 Y110. |

**42. Установка значений координат,** - G92. Инструкцию G92 можно использовать в кадре без осевой (координатной) информации или с осевой информацией. При отсутствии осевой информации все значения координат преобразуются в систему координат станка; при этом снимаются все компенсации (коррекции) и смещения нуля. При наличии осевой информации указанные значения координат становятся текущими. Инструкция G92 не инициирует каких-либо перемещений (см. рис. 2.63).

**Пример**

```
N…G92 X0 Y0        /Текущие значения координат  X и Y
                   /устанавливаются в нуль. Текущее значение
                   /координаты Z остается неизменным.
N…G92              /Снимаются коррекции и смещения нуля.
```

Инструкция G92 действует в рамках одного кадра. В том же кадре могут быть запрограммированы и другие функции; но те, которые не содержат адресов перемещений.

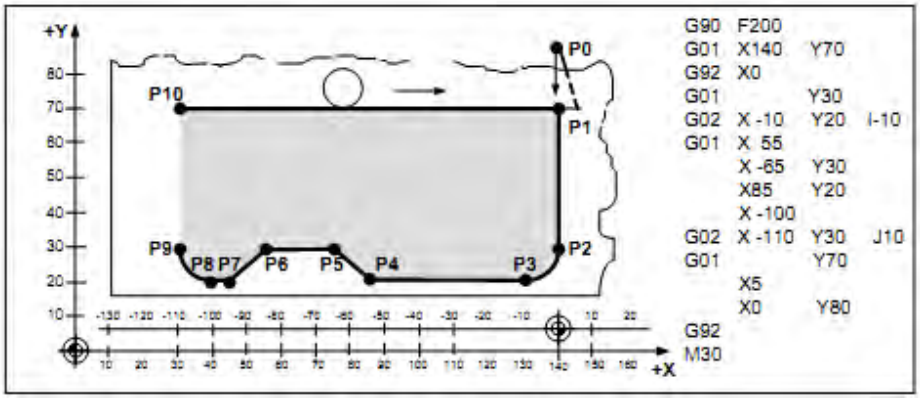



Рис.2. 63.

**43. Программирование времени,** - G93. При использовании инструкции G93 информация F-слова интерпретируется как время отработки кадра для линейной (G01) или круговой (G02, G03, G05) интерполяции. То же относится и к программированию в полярных координатах.

**Пример:**

```
N10 G93 G01 X300 Z400 A50 B120 F60   /Линейная интерполяция в кадре
                                     /осуществляется в течение 60 Сек.
```

При переходе к инструкциям G94 или G95 функция программирования времени сохраняется в памяти и восстанавливается вновь при появлении инструкции G93.

**44. Программирование подачи в мм/мин,** - G94. Система ЧПУ интерпретирует F- слово как подачу в мм/мин. Ограничения подачи определяются машинными параметрами.

**Пример:**

```
N10 G01 G94 X200 Z300 F200   /Подача равна 200 мм/мин.
N11 G04 F40                  /Выдержка времени равна 40 Сек.
N12 X300 Z400                /Подача 200 мм/мин вновь активна
```

При переходе к инструкциям G93 или G95 функция программирования подачи сохраняется в памяти и восстанавливается вновь при появлении инструкции G94.

**45. Программирование скорости (подачи, частоты вращения) с адаптацией ускорения,** - G194. Инструкция G194 позволяет ступенчато изменять активную скорость подачи в пределах кадра. Ускорение адаптируется таким образом, что заданная скорость будет достигнута только в конце кадра. Это позволяет автоматически получить «мягкое» изменение скорости. Аналогичным образом можно менять частоту вращения шпинделя для некоторых специальных траекторий. В этом случае частота вращения будет линейно изменяться вдоль траектории, а запрограммированная частота будет достигнута в конце кадра.

**Пример:**

```
N…G194 F100 X…Y…Z…        /Скорость подачи будет возрастать в пределах
                           /кадра на 100 мм/мин при каждом шаге.
N…G194 F-50 X…Y…Z…         /Скорость подачи будет убывать в пределах
                           /кадра на 50 мм/мин при каждом шаге.
N…G194…S1=100 X…Y…Z…       /Частота вращения первого шпинделя будет
                           /возрастать в пределах кадра на 100 об/мин
                           /при каждом шаге.
N…G194 F100 S2=150 X…Y…Z…  /В пределах кадра скорость подачи будет
                           /возрастать на 100 мм/мин, а частота
                           /вращения второго шпинделя будет
                           /возрастать на 150 об/мин.
```

**46. Программирование скорости подачи в мм/об,** - G95. Инструкция G95 заставляет систему ЧПУ интерпретировать последующие F-слова как подачу в мм/об.

**Пример:**

```
N9   S2000 M4              /Частота вращения шпинделя против часовой
                           /стрелки равна 2000 об/мин.
N10 G01 G95 X200 Z300 F0.2 /Скорость подачи равна 0.2 мм/об.
N11 G104 F4                /Время выстоя равно четырем оборотам
                           /шпинделя.
N12 X300 Z400              /Скорость подачи 0.2 мм/об вновь активна.
```

Инструкция G95 является модальной. Значение подачи сохраняется в памяти при переключении на инструкции G93 или G94 и становится вновь активным при повторном выборе G95.

**47. Программирование скорости резания, - G97. Поддержание постоянной скорости резания,** - G196. Инструкция G97 предполагает программирование частоты вращения шпинделя с помощью S-слова. Инструкция G196 инициирует постоянную скорость резания (в мм/мин) за счет изменения скорости той оси, которая указана в машинных параметрах.

При смене инструкций с G196 на G97, S-слово может быть опущено. В этом случае сохранится текущая частота вращения шпинделя.

**48.Установка нуля для «модульных» (modulo) осей, т.е. линейных «бесконечных осей»,** - G105. Инструкция G105 устанавливает программный нуль для условно-бесконечных осей с большими перемещениями. Для таких осей задают модуль перемещения, по достижении которого координата оси сбрасывается в нуль. Значение модуля должно быть как можно большим; например, 20 м. Система ЧПУ не допускает программирования перемещений, превышающих модуль как в положительном, так и отрицательном направлении. Пусть модуль равен 20 м., а отрицательное перемещение равно -17м. По достижении заданного перемещения, оно пересчитывается в положительное +3м. (см.рис.2.64).

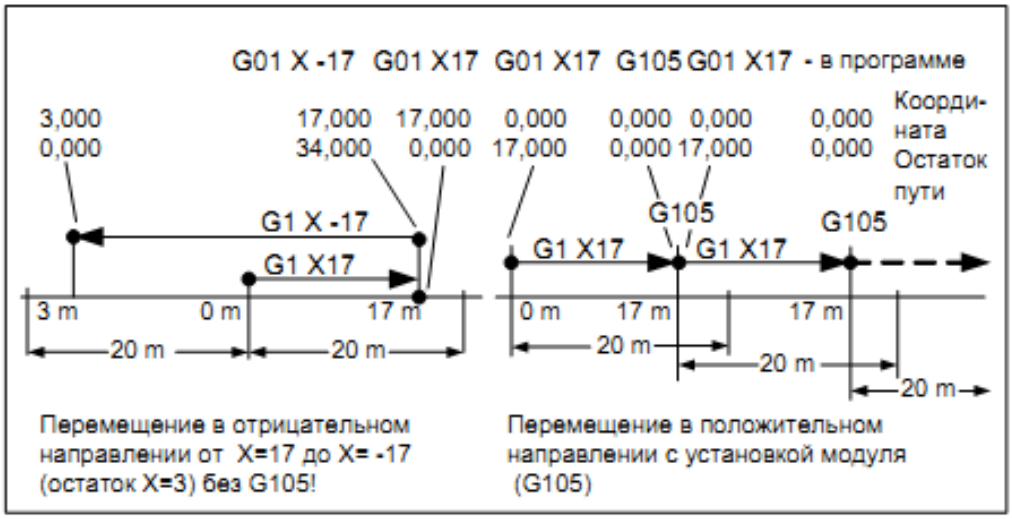



Рис. 2.64.

**Пример**

| | |
|---|---|
| N…G105 | /Установка программного нуля для всех линейных |
| | /«бесконечных осей». |
| N…G105 X200 | /Установка программного нуля для всех линейных |
| | /«бесконечных осей» и перемещение по оси X в |
| | /позицию 200 относительно нового нуля. |

**49. Опережающее управление торможением на участке перегиба контура,** - G112, G113. Цель состоит в том, чтобы на основе опережающего просмотра (Look Ahead) снизить скорость текущего кадра до такой степени, чтобы в конце следующего кадра торможение могло быть выполнено до нуля. Инструкция G112 деактивизирует опережающее управление торможением. Инструкция G113 активизирует опережающее управление торможением.

**50. Опережающее управление скоростью подачи,** - G114, G115. Приводы подачи имеют ошибку по скорости как в установившемся режиме, так и в переходных процессах. Опережающее управление скоростью подачи корректирует работу интерполятора таким образом, что ошибка по скорости уменьшается. Это позволяет более точно обрабатывать детали. Функция опережающего управления скоростью тесно связана с работой следящего привода; и далеко не все следящие приводы ориентированы на использование этой функции (ее используют следящие приводы с SERCOS-интерфейсом). Инструкция G114 активизирует опережающее управление скоростью, а инструкция G115 деактивирует опережающее управление скоростью подачи.

**Пример:**

```
N10 G114                /Опережающее управление скоростью подачи
                        /активизировано.
N20 F1000 S500
N30 G01 X1800 Y800
N160 X1500 Y1500
N170 G02 I50
N180 G114 Z0            /Опережающее управление скоростью подачи
                        /деактивировано для оси Z.
N210 G115               /Опережающее управление скоростью подачи
                        /деактивировано для всех осей.
M30
```

**51. Компенсация положения заготовки,** - G138, G139. Инструкция G138 включения компенсации положения заготовки рассоединяет координатную систему P управляющей программы и координатную систему станка M. Это позволяет адаптировать координатную систему управляющей программы к любому положению заготовки. В процессе выполнения управляющей программы все запрограммированные перемещения будут соотнесены с новой смещенной и повернутой координатной системой заготовки (см. рис.2.65).

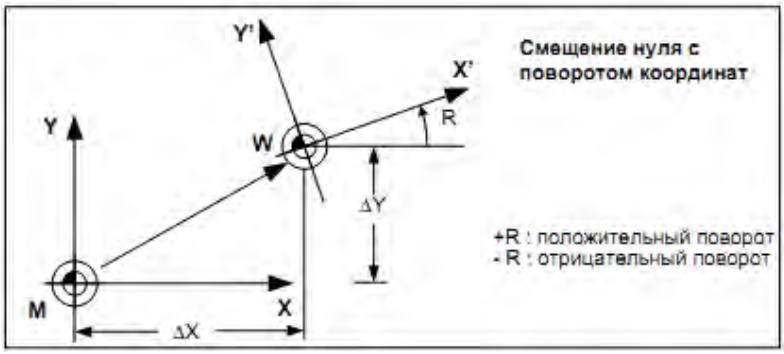



Рис.2.65.

В начале управляющей программы, в том же кадре, в каком приведена инструкция G138, программируют смещение нулевой точки W заготовки в направлениях X, Y и Z; а также и поворот осей с адресом R (угол поворота должен быть меньше 360 градусов). Все запрограммированные значения должны быть абсолютными величинами в машинной системе координат. Инструкция G139 выключает компенсацию положения заготовки.

Инструкции G138 и G139 являются модальными и взаимно уничтожают одна другую. При активной компенсации положения заготовки остаются в силе инструкции G37, G38, G54 - G59, G154-G159, G254-G259, G60, G160-G360, G168, G268, G145-G845, G147 - G847. Точно также, компенсация длины инструмента H будет принята во внимание.

**Пример:**

```
N…G90 G17 F1000 S250
N…G138 X50 Y300 Z10 R1.23      /Определение абсолютных координат
                               /нулевой точки заготовки в машинной системе
                               /координат:  X50 Y300 Z10. Поворот
                               /плоскости  X/Y против часовой стрелки  на
                               /1.23 градуса.
```

```
N…
N…
N…
N…G139                         /Выключение компенсации положения
                               /заготовки
```

**52. Внешняя коррекция инструмента,** - G145, G146, G245- G845. Речь идет об использовании одной из восьми пар коррекции инструмента, на длину и радиус. С этой целью соответствующие значения коррекции импортируются из программируемого контроллера. В результате коррекция складывается как сумма табличной (из памяти системы ЧПУ) и внешней. Инструкции первой «компенсационной группы» G145…G845 служат для включения внешней коррекции. Инструкция G146 выключает внешнюю коррекцию.

**Пример:**

```
N...G00 X0 Y0 Z0
N...H0          /Деактивизация табличной коррекции длины инструмента.
N…G146          /Деактивизация внешней коррекции инструмента.
N…G01
N…H1            /Активизация первой таблицы коррекции.
N…X10 Y10 Z10   /Перемещения с учетом первой табличной коррекции.
N…G145          /Включение внешней коррекции G145.
N…X20 Y20 Z20   /Перемещения с учетом первой табличной коррекции и
                /внешней коррекции G145.
N…G345          /Выключение внешней коррекции G145, включение
                /внешней коррекции G345
N…X30 Y30 Z30   /Перемещения с учетом первой табличной коррекции и
                /внешней коррекции G345.
N…H0            /Выключение табличной коррекции.
N…X40 Y40 Z40   /Перемещения  с учетом внешней коррекции G345.
N…G146          /Выключение внешней коррекции.
N…X0 Y0 Z0      /Перемещения без коррекции
```

**53. Внешняя коррекция инструмента с помощью второй «компенсационной группы»,** - G147, G148, G247- G847. Вторая компенсационная группа может быть использована независимо от первой и в дополнении к ней. Данные коррекции сохраняются в виде «компенсационного набора», который включает следующие параметры:

- L1, L2, L3 (коррекция длины или смещения); R (коррекция радиуса);
- SL (учет положения рабочей кромки инструмента).

С помощью параметров L1…L3 можно осуществить пространственную коррекцию инструмента, а также и одновременную коррекцию трех разных инструментов. С помощью машинных параметров, параметры L1, L2 и L3 закрепляются за осями; например, за осями X, Y, Z соответственно (см. рис.2.66). Одновременная коррекция трех разных инструментов проиллюстрирована на рис.2. 67.

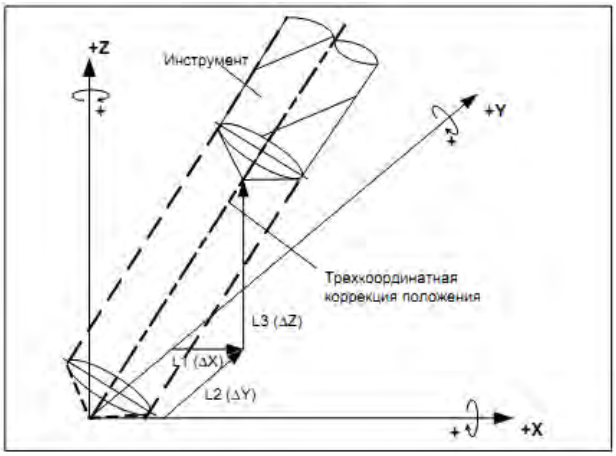



Рис. 2.66.

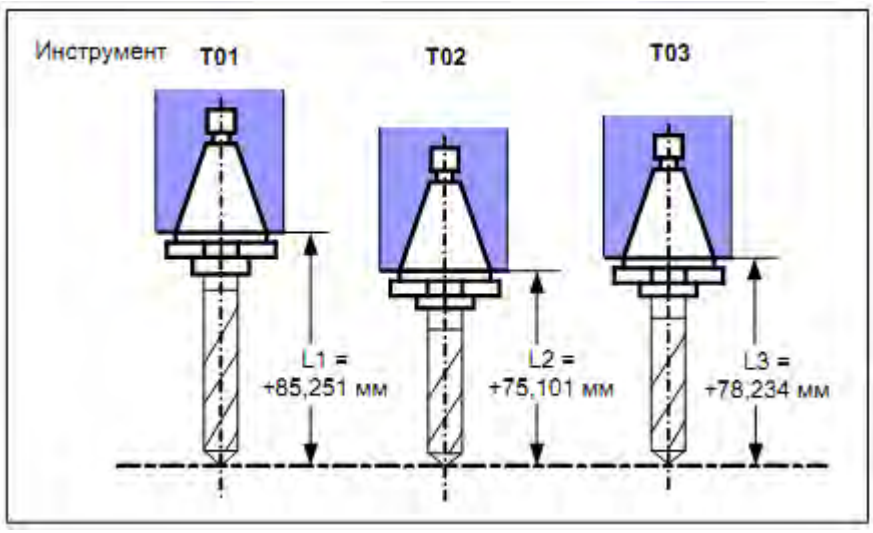



Рис.2. 67.

Параметр SL указывает на положение (ориентацию) режущей кромки инструмента (см. рис.2.68) по отношению к оси Z. Ситуация 9 на рисунке соответствует случаю, когда программным перемещением служит движение центра скругления инструмента.

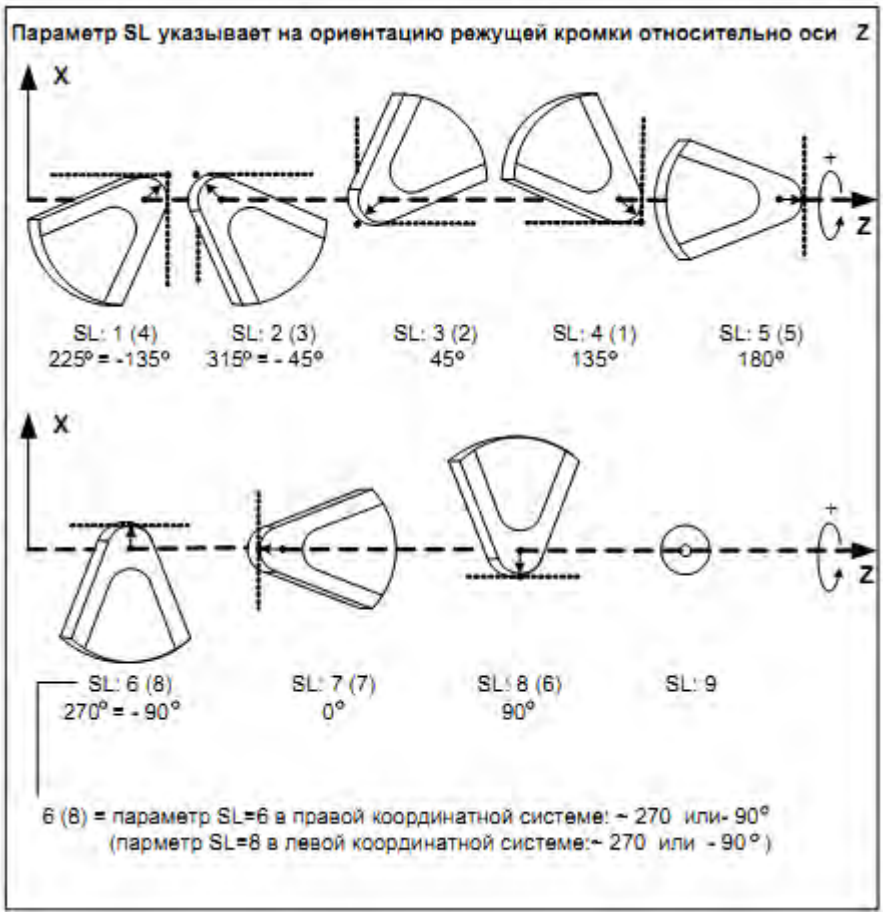



Рис. 2.68.

**54. Внешнее смещение нуля,** - G160..G360, G167. При использовании внешнего смещения нуля, все предыдущие смещения остаются в силе. Таким образом. Суммарное смещение складывается из смещений в таблицах смещения нуля ZS, сохраняемых в памяти системы ЧПУ; из активной компенсации положения заготовки; наконец, из внешнего смещения нуля. Инструкции G160, G260, G360 означают соответственно первое, второе и третье внешнее смещение нуля. Инструкция G167 отменяет внешнее смещение нуля.

**55. Точное позиционирование при ускоренном перемещении,** - G161, G162. В процессе движения следящего привода образуется динамическая «ошибка по скорости». Во всех случаях точного позиционирования, ее эффект должен быть устранен. Инструкция G161 активизирует функцию точного позиционирования специально для случая ускоренного подвода. Инструкции G164…G166 позволяют воспользоваться тремя дополнительными опциями точного позиционирования.
Инструкция G162 отменяет функцию точного позиционирования при ускоренном перемещении (см. рис.2.69).

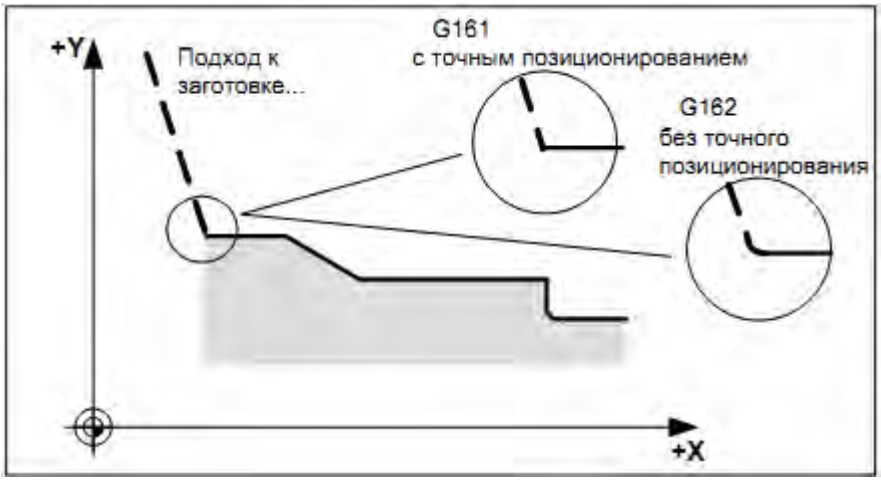



Рис. 2.69.

**Пример**

```
N10 G161        /Включение точного позиционирования без перемещений.
N11 G00 Y200    /Ускоренное перемещение с точным позиционированием.
```

Или:

```
N10 G162        /Ускоренное перемещение без точного позиционирования.
N11 G00 Y200
N50 G161 X200   /Ускоренное перемещение с точным позиционированием.
```

**56. Опции точного позиционирования,** - G164, G165, G166. Инструкция G164 инициирует снижение скорости подачи в конце кадра до нуля; при этом с помощью SERCOS-интерфейса контролируется

позиционирование всех осей в «точном окне позиционирования». Лишь после этого возможна отработка очередного кадра. Инструкция G165 инициирует снижение скорости подачи в конце кадра до нуля; при этом с помощью SERCOS-интерфейса контролируется позиционирование всех осей в «грубом окне позиционирования». Лишь после этого возможна отработка очередного кадра. Инструкция G166 инициирует снижение скорости подачи в конце кадра до нуля для всех осей; при этом никакой проверки попадания в окно позиционирования не ведется.

Для сравнения приведем различные варианты позиционирования:

- G61, - точное позиционирование при движении со скоростью подачи;
- G62, - выключение точного позиционирования со скоростью подачи;
- G161, - точное позиционирование при ускоренном перемещении;
- G162, - выключение точного позиционирования при ускоренном перемещении (только если G163 не активна);
- G163, - точное позиционирование при движении со скоростью подачи или ускоренно;
- G164, - позиционирование всех осей (V=0) в точном окне позиционирования; G165, - позиционирование всех осей (V=0) в грубом окне позиционирования;
- G166, - позиционирование всех осей (V=0) без проверки попадания в окно позиционирования.

**57. Смещение координатной системы управляющей программы, - G168, G169. Дополнительное (аддитивное) смещение управляющей программы,** - G268, G269. Все запрограммированные перемещения приводов подачи привязаны к координатной системе управляющей программы (PCS или P). Нулевая точка этой координатной системы может быть смещена по отношению к свободно выбранной нулевой точки детали (WCS or W). Смещение позволяет выполнять управляющую программу безо всякого изменения в различных подпространствах рабочего пространства станка (см. рис. 2.70).

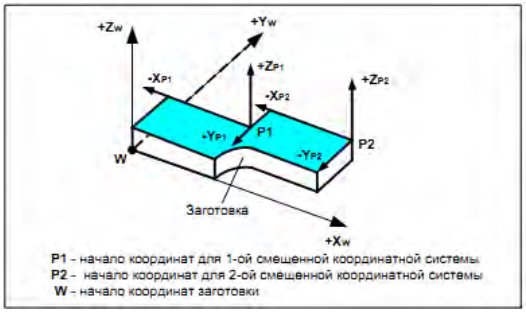



Рис.2.70.

Аддитивное смещение координатной системы позволяет последовательно выстроить несколько координатных систем; и на этой основе сконструировать управляющую программу, состоящую из одинаковых частей для обработки однообразных фрагментов детали. Все это напоминает программное смещение контура с помощью инструкции G60. Различие состоит в совместном использовании инструкций G60 и G38 (зеркальное отображение, масштабирование, поворот). Инструкция G38 не оказывает влияния на смещение координатной системы управляющей программы. Пример указанного различия показан в таблице и на рис.2.71.

| **Оси** | **Коэффициент масштабирования** | **Смещение с использованием G168,** G38 | **Смещение с использованием G60,** G38 |
|---|---|---|---|
| X | 2 | X = 1 единица | X = 2 единицы |
| Y | 2 | Y = 1 единица | Y = 2 единицы |

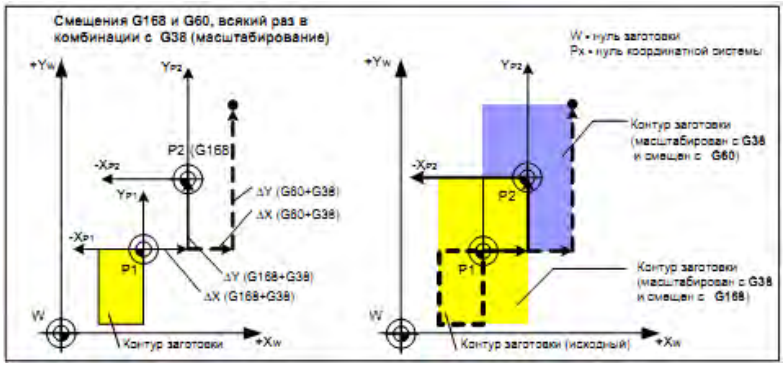



Рис.2.71.

Инструкция G168 задает смещение координатной системы управляющей программы. Инструкция G169 отменяет **все** смещения координатной системы. Инструкция G268 определяет аддитивное смещение координатной системы. Инструкция G269 отменяет **только** аддитивное смещение координатной системы управляющей программы.

**Пример:**

```
N10 G168 X10 Y10 Z50      /Установка нуля в положение  X10 Y10 Z50
                          /текущей координатной системы детали. В кадре
                          /перемещений нет.
N100 G01 X…Y…Z…           /Запрограммированные перемещения в
                          /координатной системе, введенной выше.
N110 G268 X20 Y10         /Установка нуля в положение  X30 Y20 Z50 по
                          /отношению к координатной системе детали. В
```

```
                    /кадре перемещений нет.
N200 G169           /Координатная система, введенная ранее,
                    /отменяется. Координатная система
                    /управляющей программы теперь идентична
                    /координатной системе детали.
```

**58. Смешенное программирование, абсолютное с относительным, - G190. Смешанное программирование, относительное с абсолютным, -** G191. При использовании инструкций G90 и G91 устанавливается глобальный способ интерпретации системой ЧПУ функций размерных перемещений как абсолютных или относительных. Инструкции G190 и G191 позволяют модифицировать этот способ от кадра к кадру.

Инструкция G190 устанавливает способ абсолютного программирования (по отношению к нулевой точке). Однако в последующих кадрах допускается относительное программирование (по отношению к конечной достигнутой точке) раздельно для координатных осей. Для этого к соответствующему адресу оси добавляют параметр «I» (Incremental); например «XI», «Yl», «Zl».

Инструкция G191 устанавливает способ относительного программирования (по отношению к последней достигнутой позиции) раздельно для координатных осей. Однако в последующих кадрах допускается абсолютное программирование (по отношению к текущей активной нулевой точке). Для этого к соответствующему адресу оси добавляют параметр «A» (Absolute); например, «XA», «YA», «ZA».

**Примеры**

```
N…
N20 G190                    /Начиная со следующего кадра и далее,
                            /возможно относительное программирование.
N30 G01 X100 Y150 F1000 S150 /Перемещение в позицию X100 Y150
N40 XI50 YI50                /Перемещение в позицию X150 Y200
N 50 X200 Y250               /Перемещение в позицию X200 Y250
N 60 X210 YI250              /Перемещение в позицию X210 Y500
```

```
N20 G191                    /Начиная со следующего кадра и далее,
                            /возможно абсолютное программирование;
                            /абсолютные координаты текущей позиции
                            /X10 Y10.
N30 G01 X100 Y50 F1000 S150 /Перемещение в позицию X110 Y60
N40 XA250 YA200             /Перемещение в позицию X250 Y200
N50 X200 Y250               /Перемещение в позицию X450 Y450
N60 XA100Y50                /Перемещение в позицию X100 Y500
```

**59. Ограничения частоты вращения,** - G192, G292. Верхний и нижний уровни частоты вращения могут быть установлены в пределах управляющей программы. Инструкция G192 инициирует установку в S-слове нижнего предела, а инструкция G292 - установку верхнего предела.

**Пример:**

| | |
|---|---|
| N… | |
| N100 X…Y…G192 S1500 | /Минимальная частота вращения |
| | /равна 1500 об/мин. |
| N101 X…Y…G292 S2500 | /Максимальная частота вращения |
| | /равна 2500 об/мин. |
| N…X…Y…G292 S-1 | /Отмена максимального ограничения. |
| N…X…Y…G192 S0 | /Отмена минимального ограничения |

**60. Осциллирующее движение,** - G301, G350. Осциллирующее движение может быть наложено на обычное движение любой выбранной оси в процессе линейной интерполяции группы осей. Осциллирующее движение инициируется инструкцией G301, которая является модальной в группе интерполяции (G01, G02…), см. рис.2.72. Таким образом, отмена осциллирующего движения осуществляется при появлении любой другой инструкции из этой группы. Инструкция G350 предваряет инструкцию G301, определяя основные параметры осциллирующего движения.

Синтаксис инструкции G350: OSC<осциллирующая ось> URP<верхняя точка гармоники в мм> LRP<нижняя точка гармоники в мм> F<подача в мм/мин, альтернатива OF> OF<частота осциллирующего движения, 1/Сек >. Здесь: OSC – Oscillating; URP – Upper reversing Point; LRP – Lower Reversing Point; OF – Oscillating Frequency. Синтаксис инструкции G301: X<перемещение, связанное линейной интерполяцией с Y> Y<перемещение, связанное линейной интерполяцией с X> F<подача> Time<продолжительность осциллирующего движения в кадрах, в которых перемещение отсутствует, мсек>.

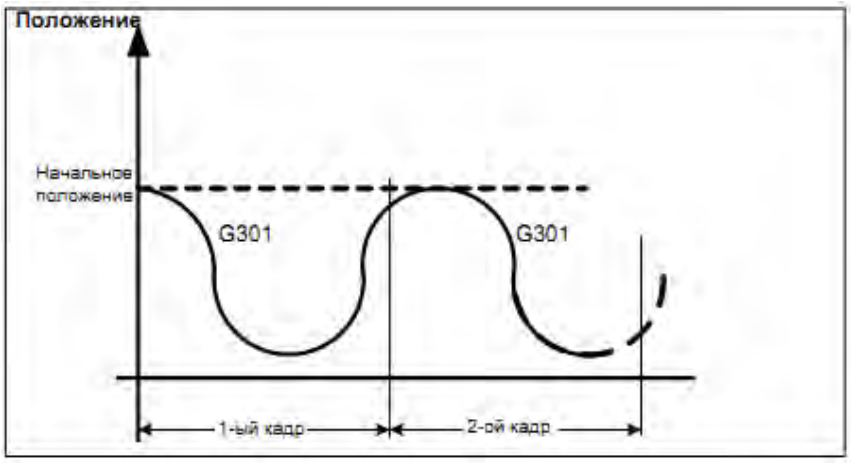



Рис.2.72.

**Пример**:

```
G350 OscX URP200 LRP100 OF5
G301 X100 Y10 F20
```

61. **Управление коллизиями, - G543, G544. Функция опережающего просмотра Look-ahead для управления коллизиями,** - G500. Инструкция G500 обнаруживает возможную коллизию при опережающем просмотре кадров с эквидистантной коррекцией; причем число кадров указывается в качестве параметра инструкции. Инструкция G543 включает управление коллизиями, а инструкция G544 выключает это управление, см. рис.2.73.

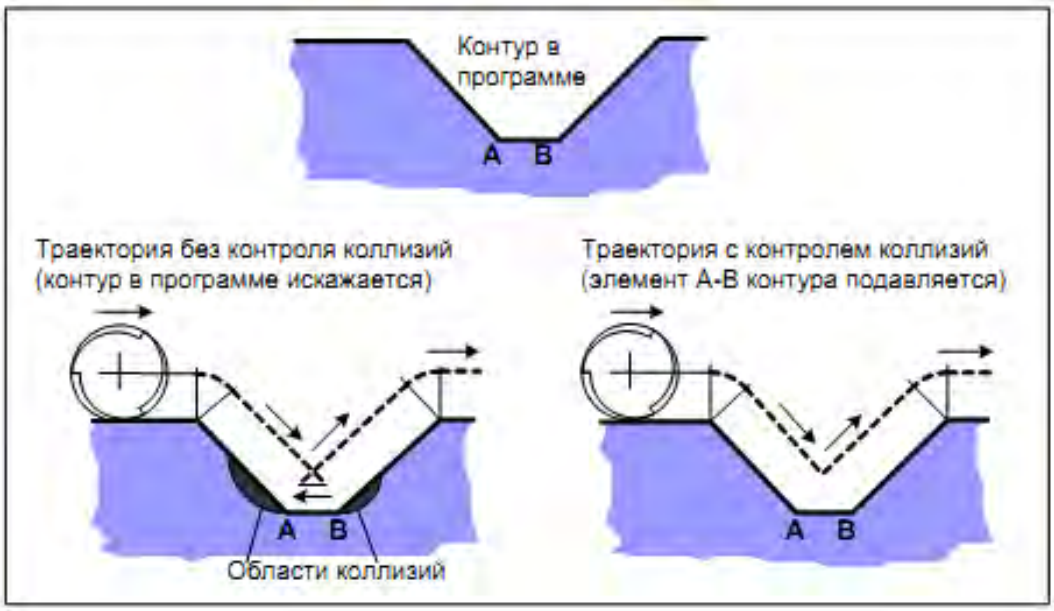



Рис.2.73.

**62. Группирование координатных осей,** - G581, G580. Группирование осей приводит к жесткому позиционному соотношению между ведущей осью и ведомыми осями. Каждая группа состоит из одной ведущей оси и до семи ведомых. Группа осей работает в одном и том же канале системы ЧПУ. За каждым каналом (в многоканальных системах ЧПУ) может быть закреплено несколько групп, см. рис.2.74. Инструкция G581служит для создания групп, а инструкция G580 предназначена для их расформирования.

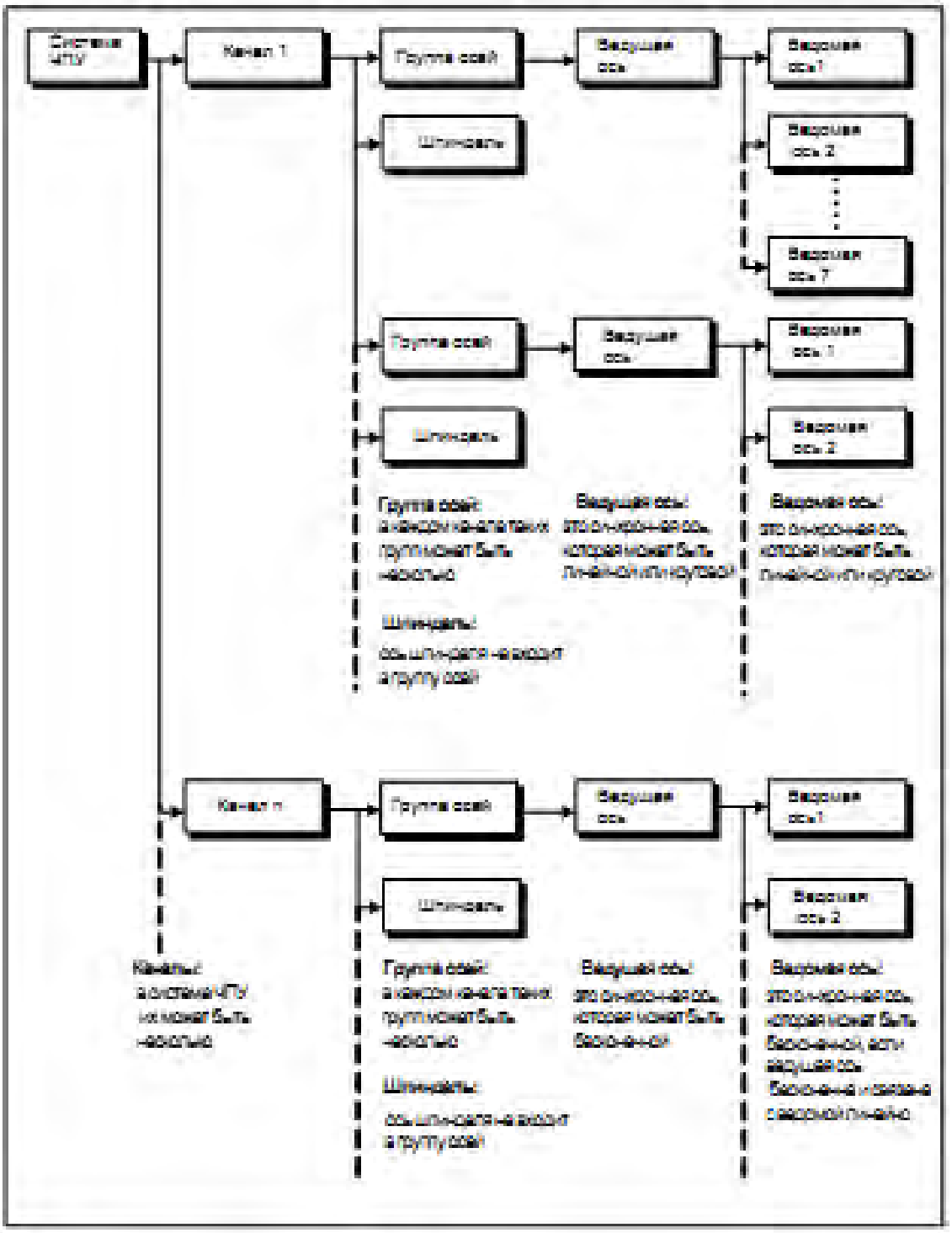


Рис. 2.74.

Существуют следующие варианты групп:

- группы с параллельными осями (например, если несколько исполнительных органов перемещаются параллельно);
- электронные гитары (с осями, которые связаны определенным передаточным отношением);
- группы с нелинейно-связанными осями.

Характеристикой группы служит отношение ведомых осей к ведущей.

Линейное отношение линейно связывает положение ведущей оси $p_m$ с положением $p_s$ ведомой оси, см. рис.2.75.

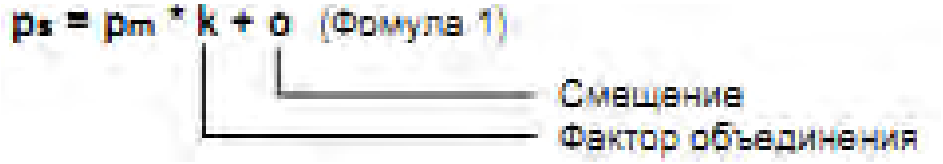



Рис.2.75.

Здесь: **к**=1 для параллельных осей; для электронных гитар значение **к** определяется настройкой электронной гитары.

Нелинейно-связанные оси представлены функцией $f(p_m)$, которая хранится в табличной форме в файловой системе системы ЧПУ, см.рис.2.76.

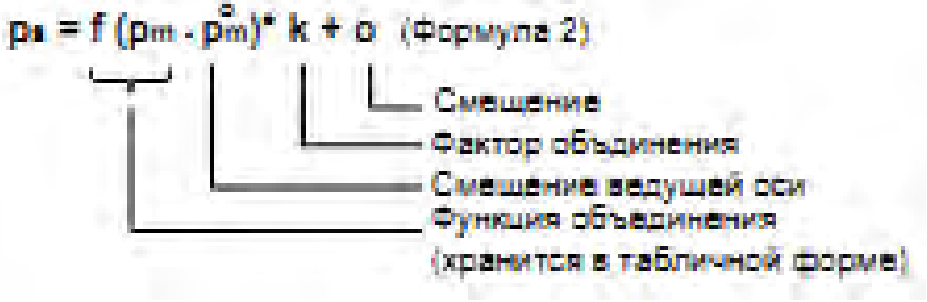



Рис.2.76.

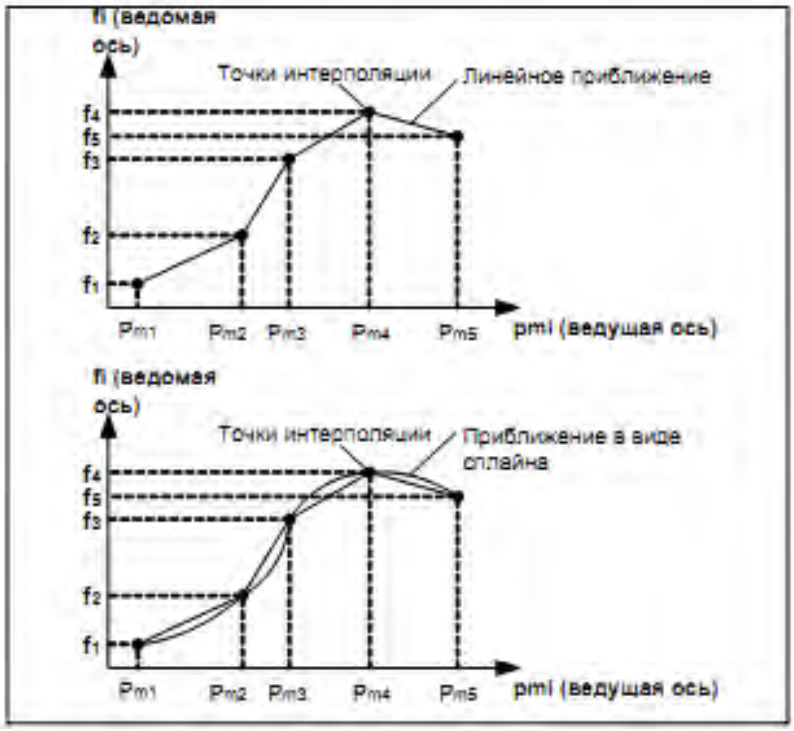



Рис.2.77.

## 2.4 Управление шпинделем

**1. Функции шпинделя**

Функции шпинделя относятся к отдельным шпинделям или шпиндельным группам. Максимальное число шпинделей равно восьми, и каждый из них может быть придан любой из четырех предусмотренных групп с помощью машинных параметров. Примеры отношений вспомогательных M-функций и шпинделей: M03 относится к первой шпиндельной группе; M103 относится к первому шпинделю; M203 относится ко второму шпинделю. Все эти вспомогательные функции включают вращение шпинделя (или шпиндельной группы) по часовой стрелке. Аналогичным образом, вспомогательные функции M13, M113, M213 включают вращение шпинделя (или шпиндельной группы) по часовой стрелке с одновременной активизацией функции охлаждения. Вспомогательные функции M04, M104, M204 включают вращение шпинделя (или шпиндельной группы) против часовой стрелки.

Аналогичным образом, вспомогательные функции M14, M114, M214 включают вращение шпинделя (или шпиндельной группы) против часовой стрелки с одновременной активизацией функции охлаждения. Вспомогательные функции M05, M105, M205 останавливают вращение шпинделя.

**2. Ориентированная остановка шпинделя (шпиндельной группы).** Вспомогательные функции M19 (для первой шпиндельной группы), M119 (для первого шпинделя), M219 (для второго шпинделя) служат для программирования ориентированной остановки вращения шпинделя (или шпиндельной группы). При этом может быть использовано или не использовано S-слово. Если S-слово не используется, то шпиндель останавливается по углу в своей относительной точке. При использовании S-слова указывают угол позиционирования в градусах по отношению к относительной точке шпинделя.

**Пример:**

```
N...M19          /Шпиндели первой группы устанавливаются соответственно в
                 /свои относительные точки.
N...M119         /Первый шпиндель устанавливается в свою относительную
                 /точку.
N...M219         /Второй шпиндель устанавливается в свою относительную
                 /точку.
N...M19 S180     /Шпиндели первой группы устанавливаются под углом 180
                 /градусов к относительной точке.
N...M119 S1=180  /Первый шпиндель устанавливается под углом 180 градусов к
                 /относительной точке.
```

**3. Использование шпиндельной бабки с зубчатыми передачами.** Общий диапазон регулирования частоты вращения шпинделя разбивается с помощью шпиндельной бабки на несколько поддиапазонов, не более четырех. Все характеристики этих поддиапазонов отражаются в машинных параметрах. Для активизации автоматического переключения в шпиндельной бабке используются вспомогательные функции M40, M140 и M240, различие между которыми такое же, как и в рассмотренных выше шпиндельных функциях.

**Пример:**

```
N...M40     /Включение автоматического выбора поддиапазона для первой
            /шпиндельной группы.
N...M140    /Включение автоматического выбора поддиапазона для первого
            /шпинделя.
N...M240    /Включение автоматического выбора поддиапазона для второго
            /шпинделя.
```

**4. Программирование частоты вращения.** Частота вращения программируется для отдельного шпинделя или для всех шпинделей группы с помощью S-слова. Варианты использования слова таковы:

- «Si=» означает, что программируется частота вращения для шпинделя номера «i»;
- «SSPGj=» означает, что программируется частота вращения для шпиндельной группы номера «j»;
- использование только лишь адреса S означает, что программируется частота вращения шпинделя той группы, которой по умолчанию принадлежит первый шпиндель.

**Пример:**

```
N...G97                          /Активизация программирования частоты
                                 /вращения шпинделя.
N...G...X...Y...Z...F...SSPG1=1000 /Частота вращения шпинделей первой группы
                                 /равна1000 об/мин.
N...G...X...Y...Z...F...S1=2000  /Частота вращения первого шпинделя равна
                                 /2000 об/мин.
N...G...X...Y...Z...F...S3=2000  /Частота вращения третьего шпинделя равна
                                 /2000 об/мин.
N...G...X...Y...Z...F...S1500    /Первый шпиндель из той группы, которой он
                                 /принадлежит по умолчанию, имеет частоту
                                 /вращения 1500 об/мин.
```

## 2.5 **Вспомогательные и специальные функции**

**1. Функция подачи с адресом F.** Функцию подачи используют для программирования относительной скорости инструмента и заготовки в процессе обработки. Система ЧПУ может интерпретировать функцию подачи по-разному, в зависимости от той или иной G-инструкции:

- как время интерполяции в секундах для инструкций G01, G02, G03 и G05 (см. инструкцию G93);
- как скорость подачи в мм/мин (см. инструкцию G94);
- как скорость подачи в мм/об (см. инструкцию G95).

**Пример:**

```
N10 G93 G01 X300 Z400 A50 B120 F60   /Запрограммировано перемещение с
                                     /линейной интерполяцией в течение 60
                                     /Сек.
```

**Пример:**

```
N10 G01 G94 X200 Z300 F200   /Скорость подачи равна 200 мм/мин.
N11 G04 F40                  /Выдержка времени равна 40 Сек.
N12 X300 Z400                /Скорость подачи 200 мм/мин вновь активна.
```

**Пример:**

```
N9  S2000 M4                 /Частота вращения шпинделя равна 2000
                             /об/мин против часовой стрелки.
N10 G01 G95 X200 Z300 F0.2   /Скорость подачи равна 0,2 мм/об
N...
N12 X300 Z400                /Скорость подачи 0,2 мм/об вновь активна.
```

**2. Функция подачи асинхронной оси с адресом FA.** Обычно асинхронные оси перемещаются ускоренно. Если по каким-то причинам этого быть не должно, то для программирования скорости подачи используют адрес FA.

**Пример:**

```
N10 G01G94 X200 Z300 F200    /Скорость подачи синхронных осей равна 200
                             /мм/мин.
N11 UA400 VA140 FA250        /Асинхронные оси UA и VA перемещаются со
                             /скоростью подачи 250 мм/мин.
```

```
N12 UA1O WA10                /Асинхронные оси UA и WA перемещаются
                             /ускоренно.
```

**3. Функция частоты вращения с адресом S, - см. раздел, посвященный программированию частоты вращения шпинделя.**

**4. Вспомогательные М-функции.** Пример использования вспомогательной М-функции для включения вращения шпинделя по часовой стрелке показан на рис.2. 78-1.

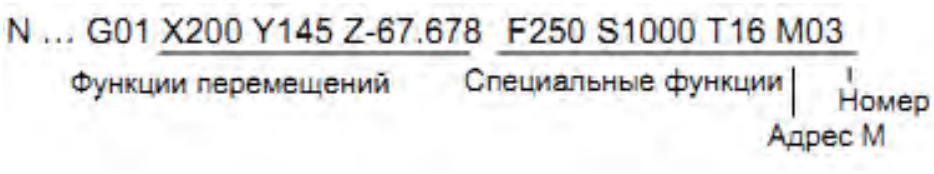



Рис. 2.78-1.

М-функции могут быть в кадре единственными; или используются в кадре вместе с другими словами (G, S, F, T).

**4.1. Вызов подпрограмм**. Наряду с различными G-инструкциями, 8 немодальных М-функций используют для вызова подпрограмм. Привязка М-функции к подпрограмме осуществляется свободно с помощью машинных параметров. При этом подпрограмма будет выполнена один раз.

**4.2. Функции останова, - M00, M01, M02, M30.** Функция М00 управляет безусловным остановом исполнения управляющей программы. Функция М01 означает условный останов управляющей программы, для выполнения которого необходимо подтверждение с панели оператора. Функции М02 или М30 используют для обозначения конца главной программы. Они могут быть использованы и в подпрограммах; при этом управление переходит к вызывающей программе.

**4.3. Вспомогательные функции, используемые при управлении шпинделем (см. раздел, посвященный программированию частоты вращения шпинделя).**

**Управление шпинделем, - M03-M219:**

- 1-ая шпиндельная группа, - M03, M04, M05, M13, M14, M19.
- 1-ый шпиндель, - M103, M104, M105, M113, M114, M119.
- 2-ой шпиндель, - M203, M204, M205, M213, M214, M219.

**Управление диапазонами вращения шпинделя,** - M40-M244:

- 1-ая шпиндельная группа, - M40, M41- M44.
- 1-ый шпиндель, - M140, M141 - M144.
- 2-ой шпиндель, - M240, M241 – M244.

**Отмена управления диапазонами вращения шпинделя, - M48-M248:**

- 1-ая шпиндельная группа, - M48.
- 1-ый шпиндель, - M148.
- 2-ой шпиндель, - M248.

**4.4. Вспомогательная функция, используемая для смены инструмента, - M06.** Вспомогательная функция М04 инициирует смену

инструмента, вызывая для этого соответствующую подпрограмму.

**5. Функция выбора инструмента с адресом Т.** С помощью этой функции запрашивают инструмент, который будет использован в очередном переходе операции. Синтаксическая структура Т-слова устанавливается машинными параметрами. Фактическая смена инструмента осуществляется с помощью вспомогательной функции М06.

**Пример:**

```
N100 T123 M06          / Выбор инструмента 123 и инициация смены
                       / с помощью вспомогательной функцией М06.

N110 G00 X100 Y200     / Начало работы с инструментом 123
N120 G01 X150 Y230
N...
N500 T234 M06          / Выбор инструмента 234 и инициация смены
                       / с помощью вспомогательной функцией М06.
```

**6. Компенсация (коррекция) инструмента**

**1. Предусловия.** Длина и радиус инструмента должны быть представлены в таблице, сохраняемой в файловой системе системы ЧПУ. Эта таблица должна быть активной.

Для компенсации длины инструмента используют Н-слово; а для компенсации радиуса инструмента используют D-слово. Для компенсации радиуса при эквидистантной коррекции должны быть активными инструкции G41 или G42.

**2. Компенсация длины инструмента (см. рис.**2.78).

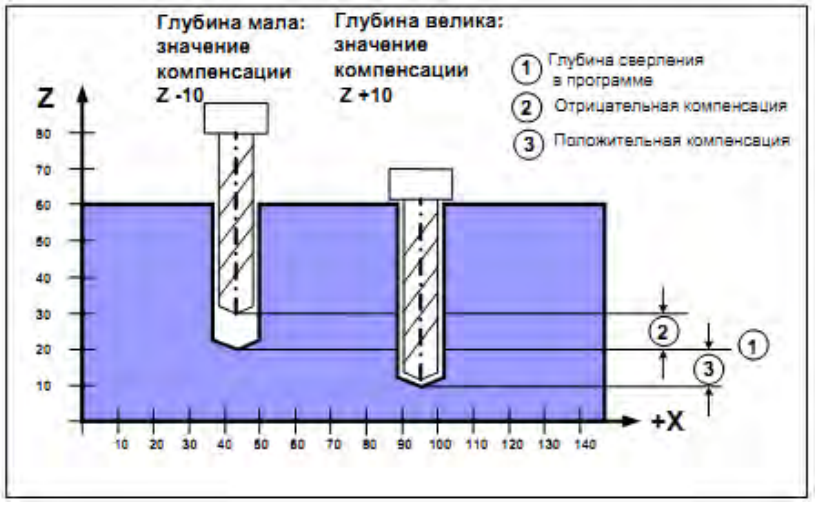



Рис. 2.78.

Функция компенсации является модальной и может быть изменена вызовом другой функции компенсации или отменена путем программирования H0. Само по себе H- слово не влечет за собой каких-либо перемещений. Пример программирования показан на рис.2.79.

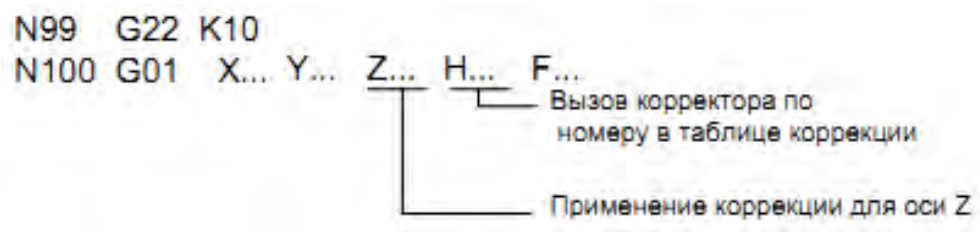



Рис.2.79.

**3. Компенсация радиуса.** Компенсация радиуса осуществляется путем вызова D-слова. Должна работать одна из инструкций, G41 или G42. С помощью одной из инструкций G17/G18/G19/G20 должна быть выбрана плоскость активной компенсации. D-слово является модальной функцией, которую заменяет новое D-слово, с новым значением компенсации. Пример программирования компенсации радиуса представлен на рис.2.80.

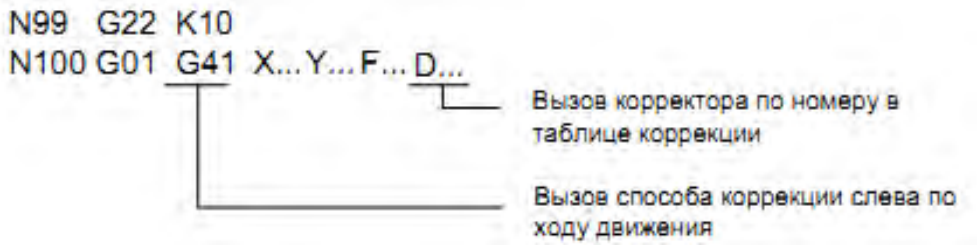



Рис.2.80.

**4. Вход в эквидистантную траекторию и выход из нее.** Если активны инструкции круговой интерполяции (G02, G03, G05), то в кадрах, инициирующих эквидистантную коррекцию (с инструкциями G41 или G42), программирование перемещений недопустимо. Инструмент входит в эквидистантную траекторию в следующем кадре (после G41 или G42) перпендикулярно к контуру. То же и для любого кадра с G41 или G42 при отсутствии в кадре запрограммированных перемещений (см. рис.2.81).

При линейной интерполяции в кадре с инструкциями G41 или G42 вход в эквидистанту осуществляется по мере движения к ее начальной точке (см. рис.2.82).

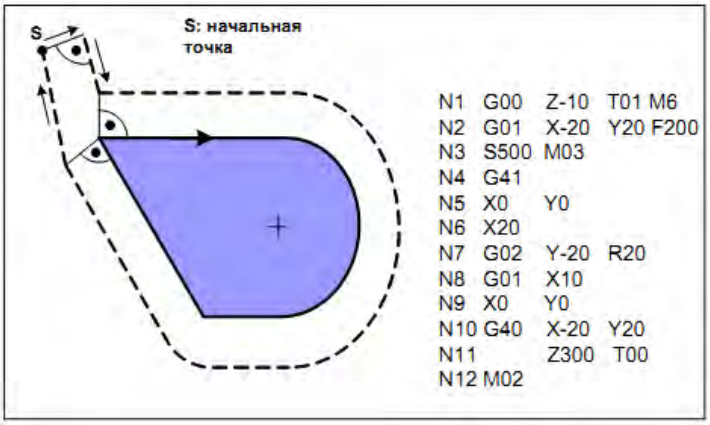



Рис.2.81.

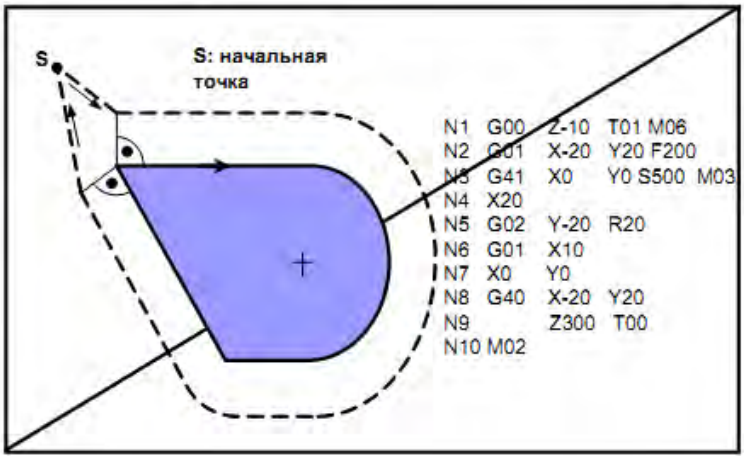



Рис.2.82.

При линейном движении к точке выхода из эквидистанты вместе с инструкцией G40 в том же кадре, эквидистантное смещение ликвидируется в процессе самого этого движения. Если в кадре с инструкцией G40 нет запрограммированных перемещений, то выход из эквидистанты осуществляется перпендикулярно к последней запрограммированной траектории (см. рис.2.83).

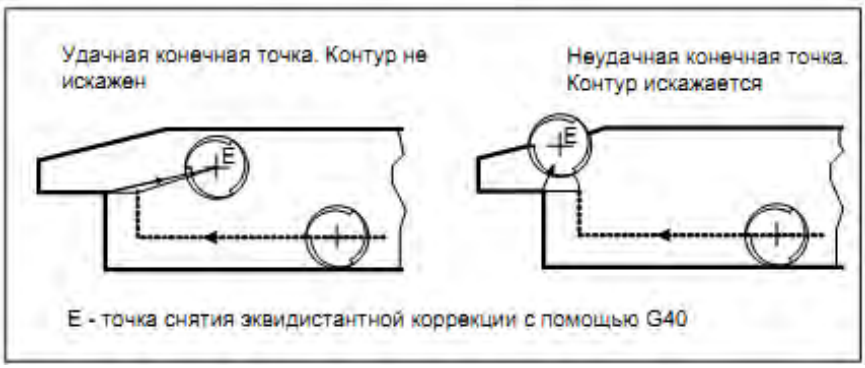



Рис.2. 83.

**5. Примеры.** Некоторые дополнительные примеры показаны: на рис.2. 84 (инструкция G41 для внешнего контура); на рис.2. 85 (инструкция G42 для внутреннего контура).

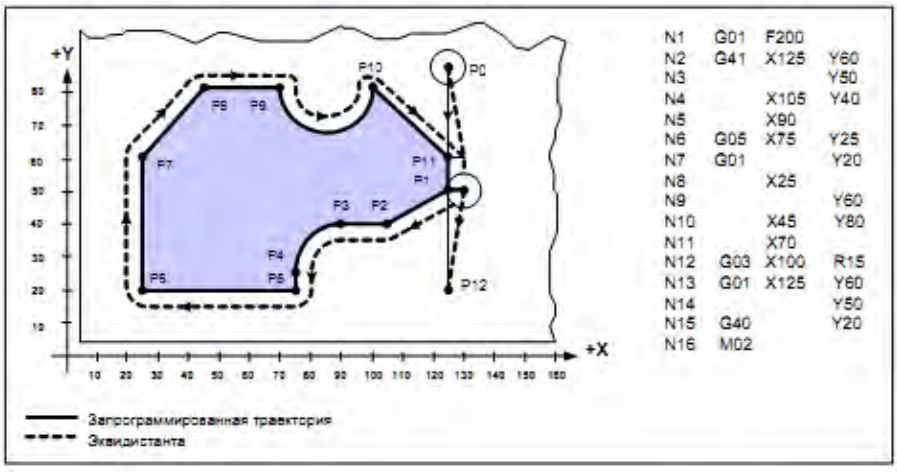



Рис. 2.84.

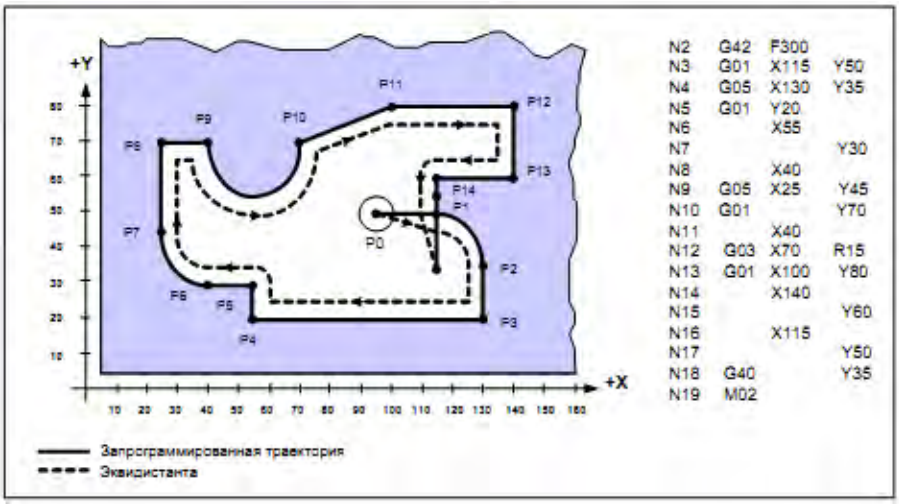



Рис.2. 85.

## 7. **Основные инструкции**

| **Инструкция** | **Описание** | **Группа** |
|---|---|---|
| G00 | Линейная интерполяция при ускоренном перемещении | 2 |
| G01 | Линейная интерполяция со скоростью подачи | 2 |
| G02 | Круговая интерполяция по часовой стрелке | 2 |
| G03 | Круговая интерполяция против часовой стрелки | 2 |
| G04 | Выдержка времени | 0 |
| G05 | Круговая интерполяция с выходом на круговую траекторию по касательной | 2 |
| G06 | Снижение допустимого уровня ускорения | 11 |
| G07 | Отмена снижения допустимого уровня ускорения | 11 |
| G08 | Управление скоростью подачи в точках перегиба | 3 |
| G09 | Отмена управления скоростью подачи в точках перегиба | 3 |
| G10 | Ускоренное перемещение в полярных координатах | 2 |
| G11 | Линейная интерполяция в полярных координатах | 2 |
| G12 | Круговая интерполяция по часовой стрелке в полярных координатах | 2 |
| G13 | Круговая интерполяция против часовой стрелки в полярных координатах | 2 |
| G14 | Возможность программирования коэффициента усиления по скорости | 9 |
| G15 | Отмена возможности программирования коэффициента усиления по скорости | 9 |
| G16 | Программирование без указания плоскости | 5 |
| G17 | Выбор плоскости X_Y | 5 |
| G18 | Выбор плоскости Z_X | 5 |
| G19 | Выбор плоскости Y_Z | 5 |
| G20 | Задание полюса и плоскости координат при программировании в полярных координатах | 5 |
| G21 | Программирование классификации осей | 0 |
| G22 | Активизация таблиц | 0 |
| G23 | Программирование условного перехода | 0 |
| G24 | Программирование безусловного перехода | 0 |
| G32 | Нарезание резьбы без компенсирующего патрона | 0 |
| G34 | Скругление двух линейных участков | 12 |
| G35 | Отмена скругления двух линейных участков | 12 |
| G36 | Восстановление параметров отклонения, установленных в машинных параметрах | 0 |
| G37 | Программирование координат полюса зеркального отображения | 22 |

| | | |
|---|---|---|
| G38 | Активизация зеркального отображения, поворота, масштабирования | 22 |
| G39 | Отмена функции зеркального отображения | 22 |
| G40 | Отмена эквидистантной коррекции | 41 |
| G41 | Эквидистантная коррекция слева по направлению подачи | 41 |
| G42 | Эквидистантная коррекция справа по направлению подачи | 41 |
| G53 | Отмена смещения нуля | 17 |
| G54… G59 | Инициация смещения нуля | 17 |
| G60 | Смещение контура в пределах координатной системы управляющей программы | 20 |
| G61 | Точное позиционирование при движении со скоростью подачи | 13 |
| G62 | Отмена точного позиционирования | 13 |
| G63 | Включение 100% от запрограммированного значения скорости | 7 |
| G64 | Привязывание скорости подачи к точке контакта фрезы и детали | 42 |
| G65 | Привязывание скорости подачи к центру фрезы | 42 |
| G66 | Активизация значения скорости, заданной потенциометром | 7 |
| G67 | Отмена смещения контура в координатной системе управляющей программы | 20 |
| G68 | Вариант сопряжения отрезков эквидистант по дуге | 43 |
| G69 | Вариант сопряжения отрезков эквидистант по траектории пересечения эквидистант | 43 |
| G70 | Программирование в дюймах | 8 |
| G71 | Отмена программирования в дюймах | 8 |
| G73 | Линейная интерполяция с точным позиционированием | 2 |
| G74 | Выход в начало координат | 0 |
| G75 | Работа с датчиком касания | 0 |
| G76 | Перемещение в точку с абсолютными координатами в системе координат станка | 0 |
| G78 | Активизация сверлильной оси | 36 |
| G79 | Деактивация одной сверлильной оси или всех сразу | 36 |
| G80 | Отмена вызова стандартных циклов | 1 |
| G81 | Стандартный цикл сверления | 1 |
| G82 | Стандартный цикл сверления | 1 |
| G83 | Стандартный цикл глубокого сверления | 1 |
| G84 | Цикл нарезания резьбы с компенсирующим патроном | 1 |
| G85 | Цикл рассверливания | 1 |
| G86 | Цикл рассверливания | 1 |
| G90 | Программирование в абсолютных координатах | 4 |
| G91 | Программирование в относительных координатах | 4 |

| | | |
|---|---|---|
| G92 | Установка значений координат | 0 |
| G93 | Программирование времени отработки кадра | 6 |
| G94 | Программирование подачи в мм/мин | 6 |
| G95 | Программирование подачи в мм/об | 6 |
| G97 | Программирование скорости резания | 35 |
| G105 | Установка нуля для линейных бесконечных осей | 0 |
| G108 | Управление подачей в точках перегиба с учетом Look Ahead | 3 |
| G112 | Деактивация опережающего управления торможением | 38 |
| G113 | Активизация опережающего управления торможением | 38 |
| G114 | Активизация опережающего управления скоростью подачи | 10 |
| G115 | Деактивация опережающего управления скоростью подачи | 10 |
| G138 | Включение компенсации положения заготовки | 23 |
| G139 | Выключение компенсации положения заготовки | 23 |
| G145…G845 | Активизация внешней коррекции инструмента со стороны программируемого контроллера | 25 |
| G146 | Выключение внешней коррекции инструмента | 25 |
| G147<br>G847 | Вторая компенсационная группа коррекции инструмента; коррекции соотнесены с осями | 52 |
| G148 | Отмена дополнительной компенсации инструмента | 52 |
| G153 | Отмена первого аддитивного смещения нуля | 18 |
| G154…G159 | Инициация первого аддитивного смещения нуля | 18 |
| G160…G360 | Внешнее смещение нуля | 24 |
| G161 | Точное позиционирование при ускоренном перемещении | 14 |
| G162 | Отмена точного позиционирования при ускоренном перемещении | 14 |
| G163 | Точное позиционирование при ускоренном перемещении и перемещении со скоростью подачи | 13 |
| G164 | Первая опция точного позиционирования | 15 |
| G165 | Вторая опция точного позиционирования | 15 |
| G166 | Третья опция точного позиционирования | 15 |
| G167 | Отмена внешнего смещения нуля | 24 |
| G168 | Смещение координатной системы управляющей программы | 46 |
| G169 | Отмена всех смещений координатной системы | 46 |
| G268 | Аддитивное смещение координатной системы управляющей программы | 47 |
| G269 | Отмена аддитивного смещения координатной системы управляющей программы | 47 |
| G184 | Цикл нарезания резьбы без компенсирующего патрона | 1 |

| | | |
|---|---|---|
| G189 | Программирование в абсолютных координатах для бесконечных осей | 4 |
| G190 | Программирование в абсолютных координатах «слово за словом» | 4 |
| G191 | Программирование в относительных координатах «слово за словом» | 4 |
| G192 | Установка нижнего предела частоты вращения в управляющей программе | 29 |
| G194 | Программирование скорости (подачи, частоты вращения) с адаптацией ускорения | 0 |
| G200 | Линейная интерполяция на ускоренном перемещении без торможения до V=0 | 2 |
| G202 | Винтовая интерполяция по часовой стрелке | 2 |
| G203 | Винтовая интерполяция против часовой стрелки | 2 |
| G206 | Активизация и сохранение в памяти максимальных значений ускорения | 0 |
| G228 | Переходы от кадра к кадру без торможения | 0 |
| G253 | Отмена второго аддитивного смещения нуля | 19 |
| G254…G259 | Инициация второго аддитивного смещения нуля | 19 |
| G292 | Установка верхнего предела частоты вращения в управляющей программе | 30 |
| G301 | Включение осциллирующего движения | 2 |
| G350 | Установка параметров осциллирующего движения | 0 |
| G408 | Формирование гладкого ускорения при движении от точки к точке | 3 |
| G500 | Обнаружение возможных коллизий при опережающем просмотре кадров | 0 |
| G543 | Включение управления коллизиями при опережающем просмотре кадров | 44 |
| G544 | Выключение управления коллизиями при опережающем просмотре кадров | 44 |
| G575 | Переключение кадров высокоскоростным внешним сигналом | 0 |
| G580 | Расформирование групп координатных осей | 48 |
| G581 | Группирование координатных осей | 48 |
| G608 | Формирование гладкого ускорения при движении от точки к точке для каждой оси в отдельности | 3 |

**Примечание:** инструкции нулевой группы немодальны.

# 3.ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

## 3.1 Архитектура микроконтроллеров AVR и PIC

В принципе, все микроконтроллеры построены по одной схеме. Система управления, состоящая из счетчика команд и схемы декодирования, выполняет считывание и декодирование команд из памяти программ, а операционное устройство отвечает за выполнение арифметических и логических операций; интерфейс ввода/вывода позволяет обмениваться данными с периферийными устройствами; и, наконец, необходимо иметь запоминающее устройство для хранения программ и данных (рис. 3.1).

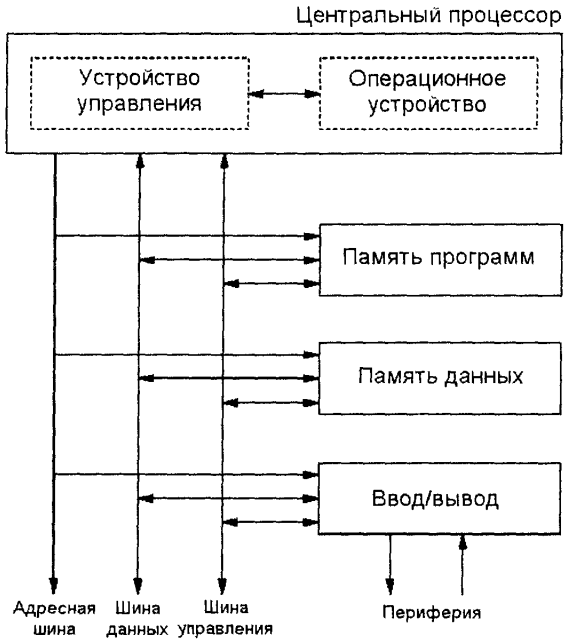



Рис. 3.1. Обобщенная структура микроконтроллера

Рассмотрим только общие для большинства микроконтроллеров AVR и PIC особенности архитектуры памяти, вопросы ввода/вывода, обработки прерываний, сброса и др.

**Память**

В микроконтроллерах AVR и PIC память реализована по Гарвардской архитектуре, что подразумевает разделение памяти команд и данных. Это означает, что обращение к командам осуществляется независимо от доступа к данным. Преимуществом такой организации является повышение скорости доступа к памяти.

К тому же, в микроконтроллерах PIC к памяти данных и к памяти команд можно обращаться фактически одновременно, что еще больше повышает скорость обработки программ. Рассмотрим, какие типы памяти могут использоваться в микроконтроллерах AVR и PIC.

***Память данных***

Память данных предназначена для записи/чтения данных, используемых программами. Является энергозависимой, то есть, при отключении питания

микроконтроллера все хранимые в ней данные, будут потеряны. В микроконтроллерах AVR память данных имеет более развитую структуру по сравнению с микроконтроллерами PIC, что показано на рис. 3.2.

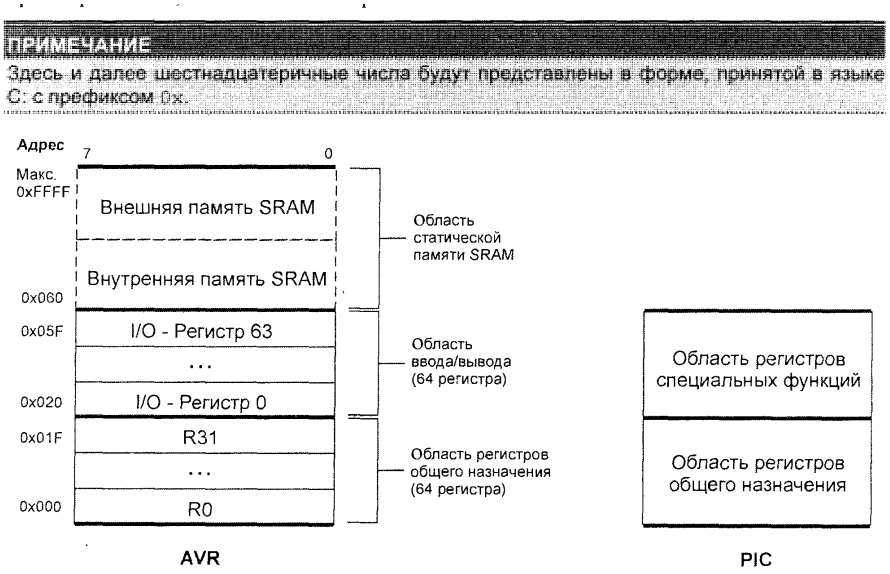



Рис. 3.2. Структура памяти данных в микроконтроллерах AVR и PIC

Область статической памяти 8Кбайт обозначена на рис. 3.2 пунктиром, поскольку используется не всеми микроконтроллерами AVR (это относится как к внутренней, так и к внешней 8Кбайт). Ее начальный адрес — 0x060, а верхний адрес — разный в различных устройствах.

В некоторых микроконтроллерах AVR можно увеличивать пространство памяти 8Кбайт посредством подключения внешних блоков памяти вплоть до 64 Кбайт, однако для этого приходится пожертвовать портами A и C, которые в этом случае применяются для передачи данных и адресов.

**Регистры общего назначения**

Область регистров общего назначения (рабочих регистров) предназначена для временного хранения переменных и указателей, используемых процессором для выполнения программ. В микроконтроллерах AVR и PIC AVR она состоит из 32 восьмиразрядных регистров (диапазон адресов 0x000 - 0x01F). В микроконтроллерах PIC регистры общего назначения также восьмиразрядные, однако их количество и диапазон адресов зависят от конкретного типа устройства.

В программах, написанных на языке C, непосредственное обращение к регистрам общего назначения обычно не требуется, если только не используются фрагменты на языке ассемблера.

**Регистры специальных функций микроконтроллеров P!C**

Регистры специальных функций используются в микроконтроллерах P1C для управления различными операциями. Как и в случае с регистрами общего назначения, их количество и адресация отличаются от устройства к устройству.

В программах, написанных на языке С, непосредственное обращение к регистрам специальных функций обычно не требуется, если только не используются фрагменты на языке ассемблера.

**Область ввода/вывода микроконтроллеров AVR и PIC**

Область ввода/вывода микроконтроллеров AVR содержит 64 регистра, используемых для управления или хранения данных периферийных устройств. К каждому из этих регистров можно обращаться по адресу ввода/вывода (начиная с 0x000) или по адресу 8Кбайт (в этом случае к адресу ввода/вывода следует прибавить 0x020). В программах на языке С обычно используются условные имена регистров ввода/вывода, а адреса имеют значение только для программ на языке ассемблера.

Имена, адреса ввода/вывода и 8Кбайт, а также краткое описание регистров из области ввода/вывода микроконтроллеров AVR. представлены в табл. 3.1. При этом следует отметить, что в различных моделях микроконтроллеров некоторые из перечисленных регистров не используются, а адреса, не указанные в табл. 3.1, зарезервированы компанией Atmel для использования в будущем.

Таблица 3.1. Описание регистров из области ввода/вывода микроконтроллеров AVR

| **Имя регистра** | **Адрес ввода/ вывода** | **Адрес** SRAM | **Описание** |
|---|---|---|---|
| ACSR | 0x08 | 0x28 | Регистр управления и состояния аналогового компаратора |
| UBRR | 0x09 | 0x29 | Регистр скорости передачи данных через UART |
| UCR | 0x0A | 0x2A | Регистр управления приемопередатчиком UART |
| USR | 0x0B | 0x2B | Регистр состояния приемопередатчика UART |
| UDR | 0x0C | 0x2C | Регистр данных приемопередатчика UART |
| SPCR | 0x0D | 0x2D | Регистр управления интерфейсом SPI |
| SPSR | 0x0E | 0x2E | Регистр состояния интерфейса SPI |
| SPDR | 0x0F | 0x2F | Регистр ввода/вывода данных интерфейса SPI |
| PIND | 0x10 | 0x30 | Выводы порта D |
| DDRD | 0x11 | 0x31 | Регистр направления передачи данных порта D |
| PORTD | 0x12 | 0x32 | Регистр данных порта D |
| PINC | 0x13 | 0x33 | Выводы порта C |
| DDRC | 0x14 | 0x34 | Регистр направления передачи данных порта C |
| PORTC | 0x15 | 0x35 | Регистр данных порта C |
| PINB | 0x16 | 0x36 | Выводы порта B |
| DDRB | 0x17 | 0x37 | Регистр направления передачи данных порта B |
| PORTB | 0x18 | 0x38 | Регистр данных порта B |
| PINA | 0x19 | 0x39 | Выводы порта A |
| DDRA | 0x1A | 0x3A | Регистр направления передачи данных порта A |
| PORTA | 0x1B | 0x3B | Регистр данных порта A |
| EECR | 0x1C | 0x3C | Регистр управления памяти EEPROM |
| EEDR | 0x1D | 0x3D | Регистр данных памяти EEPROM |
| EEARL | 0x1E | 0x3E | Регистр адреса памяти EEPROM (младший байт) |

| | | | |
|---|---|---|---|
| EEARH | 0x1F | 0x3F | Регистр адреса памяти EEPROM (старший байт) |
| WDTCR | 0x21 | 0x41 | Регистр управления сторожевым таймером |
| ICR1L | 0x24 | 0x44 | Регистр захвата таймера/счетчика T/C1 (младший байт) |
| ICR1H | 0x25 | 0x45 | Регистр захвата таймера/счетчика T/C1 (младший байт) |
| OCR1BL | 0x28 | 0x48 | Регистр сравнения B таймера T/C1 (младший байт) |
| OCR1BH | 0x29 | 0x49 | Регистр сравнения B таймера T/C1 (старший байт) |
| OCR1AL | 0x2A | 0x4A | Регистр сравнения A таймера T/C1 (младший байт) |
| OCR1AH | 0x2B | 0x4B | Регистр сравнения A таймера T/C1 (старший байт) |
| TCNT1L | 0x2C | 0x4C | Счетный регистр таймера/счетчика T/C1 (младший байт) |
| TCNT1H | 0x2D | 0x4D | Счетный регистр таймера/счетчика T/C1 (старший байт) |
| TCCR1B | 0x2E | 0x4E | Регистр управления B таймера/счетчика T/C1 |
| TCCR1A | 0x2F | 0x4F | Регистр управления A таймера/счетчика T/C1 |
| TCNT0 | 0x32 | 0x52 | Счетный регистр таймера/счетчика T/CO |
| TCCR0 | 0x33 | 0x53 | Регистр управления таймера/счетчика T/CO |
| MCUCR | 0x35 | 0x55 | Регистр управления микроконтроллером |
| TIFR | 0x38 | 0x58 | Регистр флагов прерываний от таймеров/счетчиков |
| TIMSK | 0x39 | 0x59 | Регистр маскирования прерываний от таймеров |
| GIFR | 0x3A | 0x5A | Общий регистр флагов прерываний |
| GIMSK | 0x3B | 0x5B | Общий регистр маскирования прерываний |
| SPL | 0x3D | 0x5D | Указатель стека (младший байт) |
| SPH | 0x3E | 0x5E | Указатель стека (старший байт) |
| SREG | 0x3F | 0x5F | Регистр состояния |

**Регистр состояния** SREG **микроконтроллеров AVR**

Регистр состояния содержит флаги условий микроконтроллеров AVR и располагается в области ввода/вывода по адресу $3P (адрес 8Кбайт — $F). После подачи сигнала сброса он инициализируется нулями.

В микроконтроллерах AVR для обозначения результата выполнения операций используются восемь различных флагов:

- разряд О (С) — флаг переноса (Саггу); указывает на переполнение (перенос) после выполнения арифметической или логической операции;
- разряд 1 (Z) — нулевой флаг (zero); всегда устанавливается, если результат арифметической или логической операции равен нулю; сбрасывается, если результат операции не равен нулю;
- разряд 2 (N) — флаг отрицательного результата; указывает на отрицательный результат после выполнения арифметической или логической операции;
- разряд 3 (V) - флаг переполнения при вычислениях В дополнительных кодах (Two's complement Overflow); поддерживает арифметику дополнительных кодов (арифметика кодов с дополнением до двух); устанавливается, если при выполнении соответствующей операции происходит переполнение, в противном случае _- сбрасывается;
- разряд 4 (S) - флаг знака (Sign); _- связь флагов N и V с помощью операции “Исключающее ИЛИ”; флаг знака может применяться для определения фактического результата арифметической операции;
- разряд 5 (Н) - флаг половинного переноса (Half Carry); указывает на

переполнение в младшем полубайте (разряды 0…3 байта данных); устанавливается, когда происходит перенос из младшего полубайта В старший, в противном случае - сбрасывается;

- разряд 6 (T) - флаг копирования (Transfer or Copy); предназначен для свободного применения программистом (например, в качестве буфера);
- разряд 7 (I) - общее разрешение прерываний (Global Interrupt); если прерывания, как таковые, должны быть разрешены, то должен быть установлен разряд 7 регистра состояния (в лог. 1).

**Внутренняя и внешняя память SRAM микроконтроллеров AVR**

Память SRAM микроконтроллеров AVR предназначена для хранения тех данных, которые не помещаются в рабочих регистрах, а также для организации программного стека (см. следующий подраздел). Данные обычно сохраняют в SRAM, начиная с первых адресов, а стеку соответствуют верхние адреса.

Если объема внутренней памяти SRAM недостаточно, то в некоторых микроконтроллерах AVR его можно увеличить до 64 Кбайт посредством подключения внешних блоков памяти. Для этого в регистре MCUCR (адрес В области ввода/вывоца - $35, адрес в SRAM - $55) следует установить в лог. 1 разряд SRE (разряд 7). После установки этого разряда порты A и C будут выступать в качестве шины адреса и шины данных, а выводы 7 и 6 порта D - В качестве управляющих сигналов чтения /RD и, соответственно, записи /WR для внешней памяти SRAM), независимо от того, какие направления передачи данных установлены для этих портов в соответствующих регистрах направления передачи данных.

**Стек**

Стек - это особая область памяти данных, используемая процессором для временного хранения адресов возврата из п0дпрограмм, промежуточных результатов вычислений и др. В микроконтроллерах PIC и некоторых микроконтроллерах AVR стек реализован аппаратно - для этого выделено отцельное запоминающее устройство фиксированного объема в несколько (или несколько десятков) байт. Для микроконтроллеров AVR компиляторы языка С (например, при обращении к подпрограммам) могут также создавать один или более стеков программно, начиная с верхних адресов области SRAM.

Стек действует по принципу LIFO - “Last In, First Out”, что означает “последним вошел, первым вышел”. Это означает, что новые данные вначале помещаются на вершину (первый уровень) стека, а затем, с поступлением следующих данных, “проталкиваются” на его нижние уровни. Извлечение из стека происходит В обратном порядке: вначале считываются данные, помещенные последними на вершину, после чего данные, размещенные на нижних уровнях, как бы “выталкиваются” на один уровень вверх. Ячейка памяти, которая является в данный момент вершиной стека, адресуется указателем стека (для AVR - регистровой парой SPL, SPH).

Поскольку область памяти данных, отводимая для программного стека, ограничивается только объемом памяти SRAM, при написании программ

следует следить за тем, чтобы стек не становился слишком большим, затирая полезные данные.

**Память программ**

Память программ как в микроконтроллерах AVR, так и в микроконтроллерах PIC реализована по технологии F lash-EPROM, которая ПОДразумевает программирование пользователем и вытирание электрическим способом. Размер этой памяти варьируется В зависимости от микроконтроллера и обычно составляет несколько Кбайт командных слов.

Флэш-память является энергонезависимой, то есть, сохраняет записанную в нее информацию даже после отключения питания микроконтроллера. Несмотря на то, что память этого типа - программируемая, для записи в нее используются только внешние аппаратные средства, поэтому с точки зрения программиста можно сказать, что память программ доступна только для чтения.

Адресация команд в памяти программ реализуется (: помощью специального регистра – счетчика команд, разрядность которого определяет допустимый размер этой памяти. Разрядность ячеек памяти программ, в зависимости от типа микроконтроллера, может составлять 14-16 бит.

Кроме того, следует отметить, что в микроконтроллерах PIC в первых ячейках памяти программ (начиная с адреса OxOOOO) содержатся векторы (адреса перехода) сброса и прерываний.

**Память EEPROM микроконтроллеров AVR**

Многие микроконтроллеры AVR оборудованы встроенной памятью EEPROM - электрически перезаписываемой энергонезависимой памятью. Хотя эта память и допускает запись, она редко используется для хранения программных переменных, поскольку, во-первых, медленно действующая, и, во-вторых, имеет ограниченный (хотя и довольно большой) цикл перезаписи.

Учитывая вышесказанное, память EEPROM используют, преимущественно, для хранения данных, которые не должны быть потеряны даже при потере питания. Это очень удобно, к примеру, при калибровке измерительных приборов, работающих под управлением микроконтроллеров, у которых в памяти EEPROM.

В процессе настройки сохраняются параметры корректировки. Благодаря этому, в большинстве случаев полностью отпадает необходимость в настроечных потенциометрах и триммерах.

В отличие от флэш-памяти, для записи/чтения памяти EEPROM нет необходимости в специальном программаторе - эти операции доступны программно и допускают побайтную передачу данных с помощью регистра управления EECR, регистра Данных EEDR и регистровой пары EEARL, EEARH, определяющей адрес ячейки памяти (см. табл. 3.1).

**Запись в память EEPROM**

Запись байта данных в память EEPROM осуществляется по следующей схеме:

1. Удостовериться, что в разряде EEWE (разряд 1) регистра EECR находится лог. 0 (разрешение записи).

2. Записать адрес ячейки EEPROM В регистр EEAR.

3. Записать байт данных в регистр EEDR.

4. Установить в лог. 1 разряд EEMWE (разряд 2) регистра EECR.

5. Установить в лог. 1 разряд EEWE (разряд 1) регистра EECR, чтобы активизировать процесс записи.

По окончанию цикла программирования разряд EEWE аппаратно автоматически сбрасывается в лог. 0. Программа пользователя должна непрерывно опрашивать этот разряд, ожидая появления лог. 0, прежде чем приступить к программированию следующего байта.

**Чтение из памяти EEPROM**

Чтение байта данных из памяти EEPROM осуществляется по такой схеме:

1. Записать адрес ячейки EEPROM в регистр EEAR.

2. Установить в лог. 1 разряд EERE (разрял 0) регистра EECR, чтобы активизировать процесс чтения.

3. По окончанию считывания разряда EERE аппаратное обеспечение считывает требуемый байт в регистр EEDR, после чего уже нет необходимости вновь опрашивать разряд EERE, поскольку считывание длится только один цикл такта системной синхронизации.

Перед началом операции чтения программа пользователя должна постоянно опрашивать разряд EEWE и ждать появления лог. 0. Если во время программирования памяти EEPROM В соответствующий регистр будет записан новый адрес или данные, то еще продолжающийся процесс программирования будет прерван, и результат будет неопределенным!

**Обработка прерываний**

Прерывания - это вызовы определенных функций, генерируемые, главным образом, аппаратной частью микроконтроллера. В результате прерывания выполнение программы останавливается, и происходит переход к соответствующей подпрограмме обработки прерывания.

Прерывания бывают внутренними и внешними. Источниками внутреннего прерывания являются встроенные модули микроконтроллера (например, таймер/счетчик или сторожевой таймер). Внешние прерывания вызываются сбросом (сигнал на выводе RESET) или сигналами предустановленного уровня на выводах INT. К примеру, в микроконтроллерах AVR 3а характер сигналов на выводах INTO/INTI, вызывающих прерывание, определяется с помощью разрядов регистра управления MCUCR: ISCOO (разряд 0), ISCOl (разряд 1) для входа INTO; ISClO (разряц 2), lSCl 1 (разряд 3) - для входа INTI (табл. 3.2 и табл. 3.3).

Таблица 3.2. Выбор способа активизации прерывания по входу INT0

| Разряд ISC01 | Разряд ISC00 | Описание |
|---|---|---|
| 0 | 0 | Прерывание вызывается по уровню лог. 0 на входе INT0 |
| 1 | 0 | Прерывание вызывается по ниспадающему фронту сигнала INT0 |
| 1 | 1 | Прерывание вызывается по нарастающему фронту сигнала INT0 |

Таблица 3.3. Выбор способа активизации прерывания по входу INT1

| Разряд ISC11 | Разряд ISC10 | Описание |
|---|---|---|
| 0 | 0 | Прерывание вызывается по уровню лог. 0 на входе INT1 |
| 1 | 0 | Прерывание вызывается по нарастающему фронту сигнала INT1 |
| 1 | 1 | Прерывание вызывается по ниспадающему фронту сигнала INT1 |

В ряде микроконтроллеров PIC выбор фронта для активизации прерывания по входу INT определяется состоянием разряда 6 регистра OPTION: лог. I В этом разряде соответствует прерывание по нарастающему, а лог. 0 - по ниспадающему фронту сигнала. Для установки этого разряда в языке С обычно используют специальные функции.

В микроконтроллерах AVR всем прерываниям, включая сброс, поставлен в соответствие собственный вектор прерывания - адрес в начальной области памяти программ, по которому компилятор размещает команду перехода к подпрограмме обработки прерывания. Перечень векторов прерывания в некоторых моделях микроконтроллеров AVR может выглядеть следующим образом (табл. 3.4).

Таблица 3.4. Векторы прерываний в некоторых микроконтроллерах AVR

| Адрес в памяти программ | Источник прерывания | Описание |
|---|---|---|
| 0x0000 | RESET | *Сигнал сброса* |
| 0x0001 | INT0 | Внешний запрос на прерывание по входу INT0 |
| 0x0002 | INT1 | Внешний запрос на прерывание по входу INT1 |
| 0x0003 | T/C1 | Захват по таймеру/счетчику T/C1 |
| 0x0004 | T/C1 | Совпадение с регистром сравнения A таймера T/C1 |
| 0x0005 | T/C1 | Совпадение с регистром сравнения B таймера T/C1 |
| 0x0006 | T/C1 | Переполнение таймера/счетчика T/C1 |
| 0x0007 | T/C0 | Переполнение таймера/счетчика T/C0 |
| 0x0008 | SPI | Завершение передачи данных по интерфейсу SPI |
| 0x0009 | UART | Прием байта приемопередатчиком UART завершен |
| 0x000A | UART | Регистр данных приемопередатчика UART пуст |
| 0x000B | UART | Передача данных приемопередатчиком UART завершена |
| 0x000C | ANA_COMP | Прерывание от аналогового компаратора |

В микроконтроллерах PIC источники прерывания, кроме RESET, не рассматриваются в отдельности, им обычно соответствует Один вектор, а в некоторых моделях - два вектора для прерываний с различной приоритетностью

(в микроконтроллерах AVR все прерывания имеют одинаковый приоритет, и В случае одновременного возникновения двух прерываний первым обрабатывается прерывание с меньшим номером вектора). Определять, какое именно прерывание требует обслуживания, w- задача программиста, и многие компиляторы с языка С предоставляют для этой цели готовые функции, освобождающие от необходимости самому "вычислять" источник прерывания.

В момент возникновения прерывания В стек помещается адрес возврата адрес команды, которая должна быть выполнена первой после выхода из полпрограммы обработки прерывания. В результате выполнения последней ассемблерной команды подпрограммы обработки прерывания (для микроконтроллеров AVR" это команда reti, а для микроконтроллеров PIC - ret fie) адрес возврата извлекается из стека в счетчик команд, и выполнение программы продолжается.

**Управление прерываниями в микроконтроллерах AVR**

В микроконтроллерах AVR 3а управление прерываниями отвечают, главным образом, четыре регистра:

- GIMSK (General Interrupt Mask Register) - разрешает или запрещает внешние прерывания по входу INTO/INTI;
- GIFR (General Interrupt Flag Register) - регистр флагов внешних прерываний;
- TIMSK (Timer/Counter Interrupt Mask Register) - регистр маскирования прерываний от таймера/счетчика T/CO и T/C 1;
- TIFR (Timer/Counter Interrupt Flag Register) - регистр флагов прерываний от таймеров/счетчиков.

О состоянии прерывания сигнализирует соответствующий флаг, который устанавливается или сбрасывается в регистре флагов. Даже если в регистре маски прерываний установлен соответствующий отдельный разряд разрешения прерывания, то прерывания могут активизироваться только тогда, когда в регистре состояния SREG установлен разряд общего разрешения прерываний 1 (разряд 7). Если это имеет место, и наступает прерывание, то выполнение программы ответвляется по соответствующему адресу (см. табл. 1.4) и разряд общего разрешения прерываний I В регистре SREG сбрасывается В состояние лог. 0, блокируя тем самым последующие прерывания. Если требуется прервать подпрограмму другим прерыванием, то после входа в подпрограмму обработки прерывания программа пользователя должна установить флаг 1 в лог. 1.

Вместе с входом в подпрограмму обработки прерывания аппаратно сбрасывается также и соответствующий флаг, вызвавший прерывание. Некоторые флаги прерываний могут быть сброшены самим пользователем посредством установки соответствующего флага в лог. 1.

**Регистр GIMSK**

Регистр GIMSK (рис. 3.3), расположенный в области ввода/вывода по адресу 0x0038 (адрес в SRAM - 0xOOSB0058), используется для разрешения внешних прерываний.

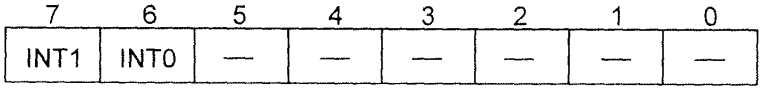



Рис. 3.3. Структура регистра GIMSK микроконтроллеров AVR

Если разряд INTI/INTO установлен в лог.1, то внешнее прерывание по входу INTI/INTO будет разрешено до тех пор, пока установлен в лог. 1 разряд 1 в регистре состояния SREG.

**Регистр GIFR**

Состояние внешнего прерывания определяется по регистру GIFR (рис. 3.4), который расположен В области ввода/вывода по адресу 0x0028 (адрес SRAM 0X0059).

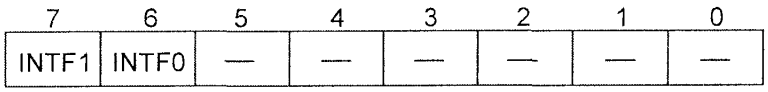



Рис. 3.4. Структура регистра GIFR микроконтроллеров AVR

Флаг INTF l/INTFO устанавливается в лог. 1, если возникает внешнее прерывание по сигналу на выводе lNTl/INTO. При входе в подпрограмму обработки прерывания этот разряд переводится аппаратно в исх0дное состояние лог. О.

**Регистры TIMSK M TIFR**

Регистр TIMSK (рис. 3.5), расположенный в области ввода/вывода по адресу охо039 (адрес в SRAM - охо059), используется для разрешения прерываний от таймеров/счетчиков (рассматриваются ниже в этой же главе).

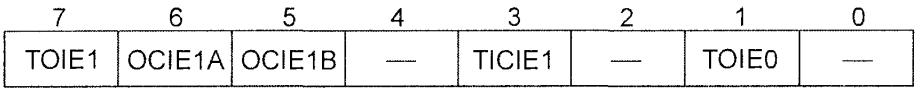



Рис. 3.5. Структура регистра TIMSK микроконтроллеров AVR

Состояние прерываний, имеющих отношение к таймерам/счетчикам микроконтроллеров AVR, определяется по регистру TlFR (рис. 3.6), который расположен в области ввода/вывоца по адресу 0x0038 (адрес SRAM - 0x0058).

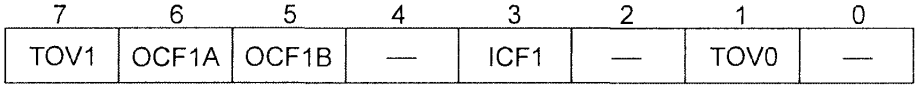



Рис. 3.6. Структура регистра TIFR микроконтроллеров AVR

Когда разряд TO1E1 и разряд 1 в регистре состояния SREG установлены В лог. 1, то разрешено прерывание при переполнении T/C1. В случае переполнения в регистре TIFR устанавливается флаг TOVI.

Если разряд OCFIA и разряд 1 в регистре состояния SREG установлены в лог. 1, то разрешено прерывание при совпадении 00держимого регистра сравнения А с текущим состоянием T/C1. В случае совпадения, в регистре TIF R устанавливается флаг OCFI A.

Если разряд OC1E1B и разряд I В регистре состояния SREG установлены В лог. 1, то разрешается прерывание при совпадении содержимого регистра сравнения В с текущим состоянием T/C1. В случае совпадения, в регистре TIFR устанавливается флаг OCFIB.

Если разряд T1C1E1 и разряд 1 в регистре состояния SREG установлены В лог. 1, то разрешается прерывание при выполнении условия захвата. Когда возникает срабатывание по захвату, В регистре TIF R устанавливается флаг ICF 1.

Если разряд TOIEO и разряд I в регистре состояния SREG установлены В лог. 1, то разрешается прерывание при переполнении таймера/счетчика T/CO. В таком случае, В регистре TIF R устанавливается флаг TOVO.

Установка в лог. I одного из флагов в регистре TIFR приводит к переходу по соответствующему вектору прерывания. При входе в подпрограмму обработки прерывания, флаг в регистре TIFR аппаратно сбрасывается в лог. 0.

**Управление прерываниями в микроконтроллерах** PIC

В микроконтроллерах PIC управление прерываниями реализовано с помощью регистров специальных функций, и отличается от устройства к устройству. К примеру, в микроконтроллерах P1C12C6x, PICI4000, PICI6x для этой цели используются регистры INTCON (рис. 3.7), PIE и PIR, а программы обработки прерываний всегда начинают исполняться с адреса oxo04.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GIE | PEIE | TOIE | INTE | RBIE | TOIF | INTF | RBIF |

Рис. 3.7. Регистр INTCON микроконтроллеров PiC12Cбx, РЮ14ООО, PlC16x

Разряд GIE - это флаг общего разрешения прерываний. Если он установлен в лог. I, то все немаскированные прерывания разрешены, если же он сброшен в лог. 0, то все прерывания запрещены.

Разряд PEIE регистра INTCON может использоваться в качестве флага разрешения всех прерываний от периферии, определяемых с помощью регистров PIE и PIR.

Флаг TOIE разрешает (лог. 1) или запрещает (лог. О) прерывания при переполнении таймера TMR0), а флаг TOIF определяет запрос на соответствующее прерывание.

Разряд INTE - флаг разрешения внешнего прерывания по входу INT, а разряд INTF - флаг запроса на прерывания по этому входу. Аналогичное значение, но для порта B имеют разряды RBIE и RBIF.

Регистр PIE c0держит флаги разрешения прерываний от периферийных устройств, а регистр PIR - соответствующие флаги запросов на прерывание. Позиции разрядов в этих регистрах для различных микроконтроллеров отличаются.

**Таймеры/счетчики микроконтроллеров AVR**

В микроконтроллерах AVR могут использоваться следующие таймеры/счетчики:

- 8- или 16-разрядный T/CO;
- 16-разрядный T/C1;
- 8- или 16-разряцный T/C2.

Регистры управления В этом случае называются TCCRO, TCCRI и TCCR2 (расположены В области вода/вывода), а режим работы и коэффициент деления частоты осциллятора определяется с помощью разрядов CSx2, CSx1 и CSx0 этих регистров. К примеру, для таймеров/счетчиков T/CO и T/C1 выбор режима и входного такта можно определить с помощью комбинаций разрядов, представленных в табл. 3.5.

Таблица 3.5. Выбор режима и входного такта для T/C0 и T/C1 микроконтроллеров AVR

| CSx2 | CSx1 | CSx0 | Описание |
|---|---|---|---|
| 0 | 0 | 0 | Останов |
| 0 | 0 | 1 | Режим “Таймер”, такт = такт системной синхронизации |
| 0 | 1 | 0 | Режим “Таймер”, такт = такт системной синхронизации / 8 |
| 0 | 1 | 1 | Режим “Таймер”, такт = такт системной синхронизации / 64 |
| 1 | 0 | 0 | Режим “Таймер”, такт = такт системной синхронизации / 256 |
| 1 | 0 | 1 | Режим “Таймер”, такт = такт системной синхронизации / 1024 |
| 1 | 1 | 0 | Режим “Счетчик”, такт — внешний на входе T0 (T1), активный фронт сигнала — ниспадающий |
| 1 | 1 | 1 | Режим “Счетчик”, такт — внешний на входе T0 (T1), активный фронт сигнала — нарастающий |

Для T/C2 комбинации разрядов CS22, CS21 и CS20 могут иметь разное значение для различных моделей микроконтроллеров.

T/CO

Схема работы таймера/счетчика T/CO, представлена на рис. 3.8.

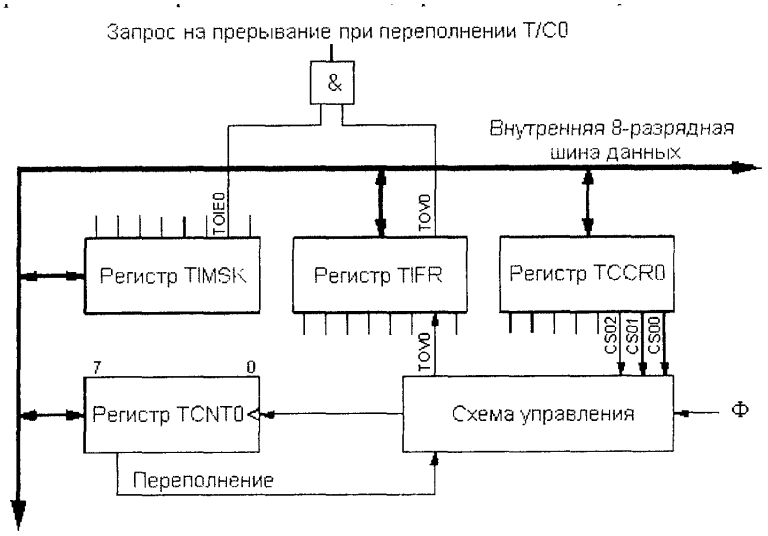



Рис. 3.8. Схема таймера/счетчика T/CO

Как только с помощью разрядов CS00, CS01 и CS02 регистра TCCRO (адрес 0x33 В области ввода/вывоца, адрес 0x53 В SRAM) Для делителя частоты будет установлена комбинация, отличная от 000, таймер/счетчик T/CO по каждому импульсу, поступающему на тактовый вхол, начинает увеличивать на единицу содержимое регистра TCNTO (адрес ox32 В области ввода/вывода, адрес 0x52 В SRAM). Когда состояние счетчика В регистре TCNTO изменяется с OxFF на OxOO, В регистре TIFR (адрес Ox38 В области ввода/вывода) устанавливается флаг переполнения TOVO.

Таймер/счетчик T/CO хорошо подходит для оценки временных интервалов. Для этого В x0де выполнения программы В регистр TCNTO‘: записывается исходное значение. Затем может быть запущен T/C0 с требуемым входным тактом. Программа ожидает появления В регистре TIFR флага переполнения TOVO, указывающего на то, что требуемое время истекло.

Предположим, частота системной синхронизации составляет 4 МГц, а некоторое действие должно выполняться программой каждые 0,5 с. В этом случае можно воспользоваться делением частоты на 8, что соответствует частоте тактирования 500 кГц или 2 мкс. Таким образом, на подсчет 256 тактовых импульсов счетчику потребуется 512 мкс. Это значение должно быть кратно 500 мкс, чтобы с помощью множителя 1000 в программе можно было реализовать требуемое действие в точности с периодом 500 мс. Для этого В счетчик перед началом каждого счета должно быть загружено значение 6, чтобы До переполнения выполнялся подсчет не 256, а только 250 тактовых импульсов.

## T/C1

16-разрядный таймер/счетчик T/C1 гораздо сложнее T/CO (рис. 3.9).

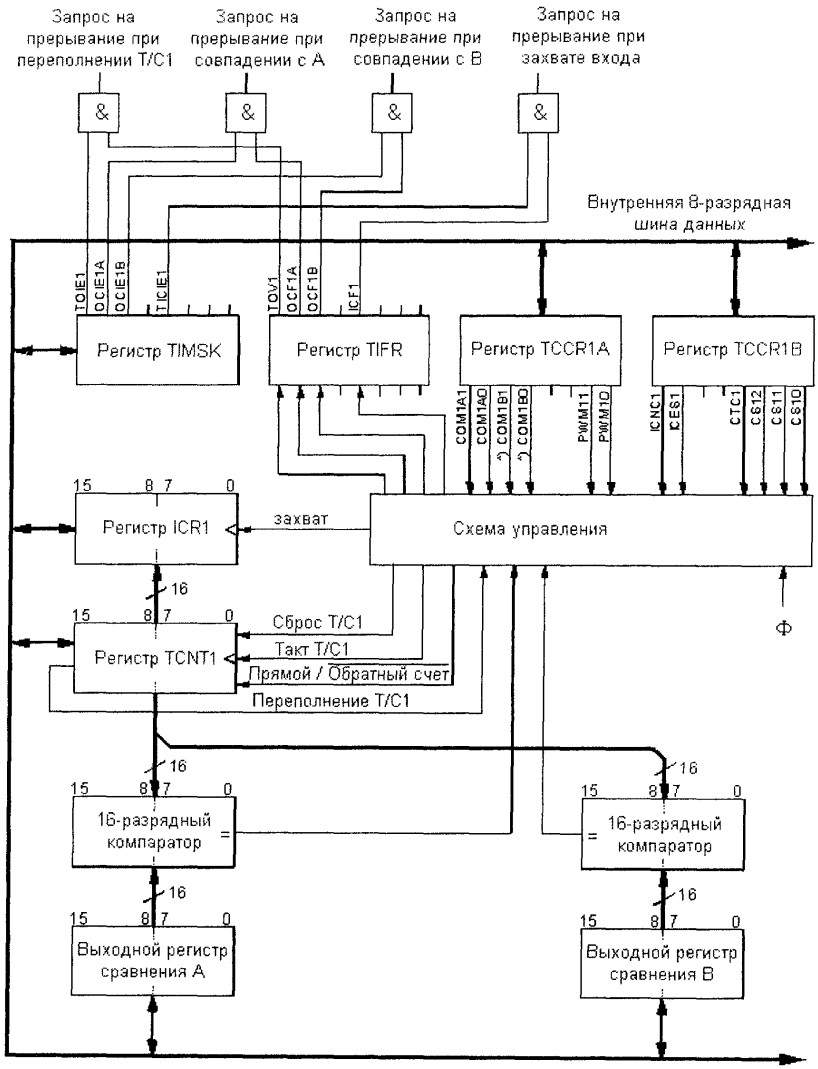



Рис. 3.9. Схема таймера-счетчика T/C1

Рассмотрим назначение отдельных регистров:

- TCNT1 - счетный регистр (содержимое счетчика);
- TCCR1A - регистр управления для определения реакции выводов
- OC1A/OC1 B в случае совпадения состояния счетчика в регистре TCNT1c регистрами сравнения OCRIA/OCRIB, а также для выбора режима широтно-импульсной модуляции;
- TCCR1 B - регистр управления для настройки делителя частоты, для разрешения подачи сигнала сброса для регистра TCNT1 и для управления захватом;
- ICR1 - регистр захвата по входу (при появлении на выводе ICP фронта входного сигнала, определенного как активный, текущее состояние счетчика будет перенесено в этот регистр);
- OCR1A, OCR1B - регистры сравнения; их c0держимое постоянно сравнивается с состоянием счетчика. В случае совпадения выполняются действия, определенные регистром TCCRIA.

Регистр управления TCCRIA (рис. 3.10) находится в области ввода/вывода по адресу Ox2F (адрес Ox4F B SRAM).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| COM1A1 | COM1A0 | COM1B1 | COM1B0 | — | — | PWM11 | PWM10 |

Рис. 3.10. Регистр TCCR1A таймера/счетчика T/C1

Разряды COM1A1/COM1A0 и COM 1B1/COM1Bo определяют состояние вывода OC1A/OC1B при совпадении содержимого регистра сравнения A/B с содержимым счетчика. Возможные настройки для режима сравнения показаны в табл. 3.6.

Таблица 3.6. Возможные варианты для работы в режиме сравнения

| COM1x1 | COM1x0 | Действия в случае совпадения |
|---|---|---|
| 0 | 0 | Выходное значение отсутствует |
| 0 | 1 | При совпадении OC1x переключается в другое состояние |
| 1 | 0 | При совпадении на выходе OC1x устанавливает лог. 0 |
| 1 | 1 | При совпадении на выходе OC1x устанавливает лог. 1 |

В случае активизации режима ШИМ, разряды 4-7 в регистре TCCRIA имеют значения, отличные от указанных в табл. 3.6. Когда регистр управления TCCRIA определяет работу в конфигурации широтно-импульсного модулятора, то T/C1 работает как суммирующий и вычитающий счетчик, осуществляя циклические переходы от OxOOOO к максимальному значению TOP, и затем снова возвращаясь к OxOOOO. При запрограммированной разрешающей способности ШИМ в N разрядов значение TOP рассчитывается как

$$TOP = 2^N - 1.$$

Частота fшим, с которой повторяются циклы ШИМ, вычисляется по формуле:

$$f_{PWM} = f_{T/C1} / (2^{N+1} - 2),$$

причем частота таймера/счетчика $f_{T/c1}$ выбирается с помощью разрядов CS10-CS12 регистра TCCRIB, а разрешающая способность N - с помощью разрядов PWM10 и PWMI 1 регистра TCCR1A. Соответствующие взаимосвязи показаны в табл. 3.7.

Таблица 3.7. Выбор режима ШИМ с помощью разрядов PWM11 и PWM10

| PWM11 | PWM10 | Разрешающая способность | Значение TOP | Частота ШИМ |
|---|---|---|---|---|
| 0 | 0 | Режим ШИМ не активен | | |
| 0 | 1 | 8 разрядов | 0x00FF (255) | $f_{T/C1}$ / 510 |
| 1 | 0 | 9 разрядов | 0x01FF (511) | $f_{T/C1}$ / 1022 |
| 1 | 1 | 10 разрядов | 0x03FF (1023) | $f_{T/C1}$ / 2046 |

Когда состояние счетчика В регистре TCNT1 совпадает со значением 10 младших разрядов регистра OCRlA/OCRIB, то, В зависимости от состояния разрядов COM1A1/COM1A0 или COM1B1/COM1B0 регистра TCCRIA, вывод OC1A/OC1B последующим тактовым импульсом устанавливается или сбрасывается. Соответствующие взаимосвязи показаны в табл. 3.8.

Таблица 3.8. Возможности выбора для режима сравнения при работе ШИМ

| COM1x1 | COM1x0 | Действие в случае совпадения |
|---|---|---|
| 0 | 0 | На выводе OC1x нет никакого сигнала |
| 0 | 1 | На выводе OC1x нет никакого сигнала |
| 1 | 0 | Неинвертирующий широтно-импульсный модулятор. В случае соответствия, при суммирующем подсчете на выводе OC1x устанавливается лог. 0, а при подсчете с вычитанием — лог. 1 |
| 1 | 1 | Инвертирующий широтно-импульсный модулятор. В случае соответствия, при суммирующем подсчете на выводе OC1x устанавливается лог. 1, а при подсчете с вычитанием — лог. 0 |

В случае неинвертирующего щиротно-импульсного модулятора, коэффициент заполнения g прямоугольного сигнала на выводе с ШИМ соответствует значению п/ (2N - 1), где n - значение в соответствующем регистре OCR, а N - разрешающая способность ШИМ в разрядах (рис. 3.10).

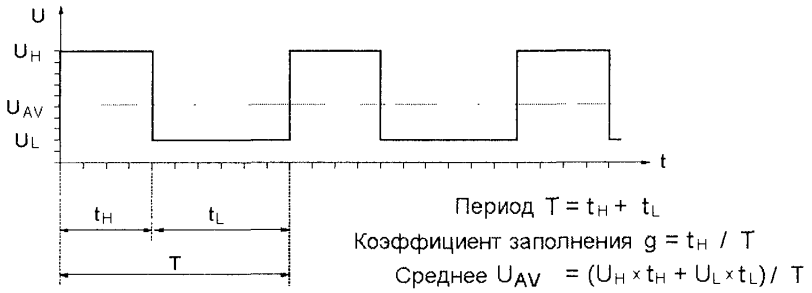



Рис. 3.10. Определение периода Т, коэффициента заполнения 9 и среднего арифметического UAV прямоугольных импульсов напряжения U

Если регистр сравнения OCRlA/OCRIB содержит значение TOP или О, то на соответствующем выводе, В соответствии с правилами, представленными В табл. в.9, постоянно поддерживаются уровень лог. 0 или лог. 1

На рис. 3.11 на примере трехразрядной ШИМ показано формирование неинвертированного и инвертированного выходного ШИМ-сигнала на выходе OC1B. На диаграмме А показан примерный вид ступенчатого сигнала, соответствующий состоянию счетчика TCNTI, На диаграмме В - неинвертированный, а на диаграмме С - инвертированный выходной сигнал. Продолжительность периода TPWM В этом случае вычисляется в соответствии с рассмотренным выше уравнением Тр…м = ТТ,… - (2NH - 2). Таким образом, при N: 3 период ШИМ-сигнала состоит из 14 периодов тактового сигнала fT/m На входе TCNT1.

Таблица 3.9. Вывод ШИМ для особых случаев OCR1x = TOP или OCR1x = 0

| COM1x1 | COM1x0 | OCR1x | Вывод OC1x |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | TOP | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | TOP | 0 |

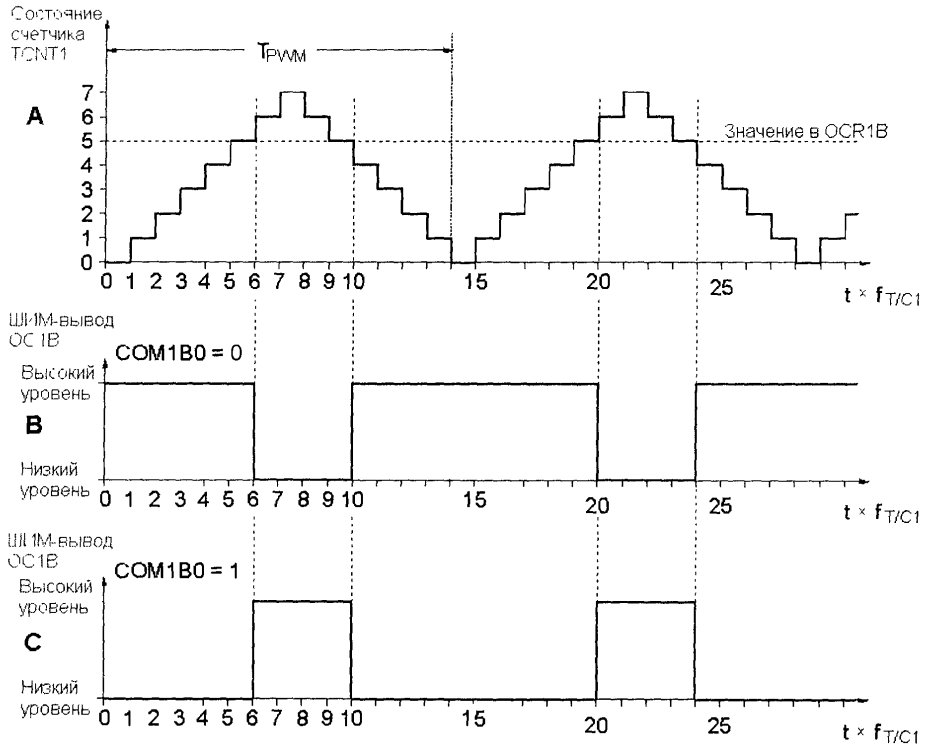



Рис. 3.11. Способ формирования неинвертированных и инвертированных выходных ШИМ-сигналов

В данном примере регистр сравнения OCRlB содержит значение 5. В регистре TCNTI, учитывая тот факт, что его исХОДное значение равно О, значение 5 появляется после пяти тактовых импульсов. На следующем тактовом импульсе, после распознания совпадения на выводе OC1B устанавливается уровень лог. 0 (рис. 3.11, В).

Регистр TCNTl инкрементируется далее до тех пор, пока не будет достигнуто значение TOP, которое при трехразрядной ШИМ составляет 7. Как только достигнуто значение TOP, направление счета меняется на обратное, и регистр выполняет вычитание.

Регистр TCNTl декрементируется далее до тех пор, пока опять не будет достигнуто значение 0. Это происходит после в общей сложности четырнадцати тактовых импульсов, считая от начального значения 0. Таким образом завершается период ШИМ-сигнала, направление счета вновь меняется на обратное и регистр TCNTl опять выполняет сложение. Как видно на рис. 3.11 (В), “высокая” составляющая выходного сигнала составляет 6 тактовых периодов, а “низкая” - 4. Таким образом, коэффициент заполнения g = 6/10 или g = 3/5.

Аналогично, диаграмма С на рис. 3.11 показывает соотношения для инвертированного выходного ШИМ-сигнала.

В режиме ШИМ устанавливается флаг переполнения TOV] , если счетчик при достижении состояния 0 меняет направление счета на обратное. Это прерывание по Т/С1 при переполнении, как и при нормальной работе В режиме счетчика, вызывается В том случае, если установлен флаг общего разрешения прерываний 1.

В регистре состояния SREG, а также флаг T01E1 в регистре TIMSK. В соответствии с этим, прерывания при совпадении регистров TCNTI и OCRlA/OCRIB вызываются тогда, когда в регистре TIMSK установлен флаг общего разрешения прерываний и флаг OCIElA/OCIEIB.

В отношении таймера/счетчика Т/С1 осталось рассмотреть еще регистр управления TCCRIB (адресу Ox2E В области ввода/вывода, адрес Ox4E в SRAM). Структура рег'истра TCCRIB показана на рис. 3.12.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ICNC1 | ICES1 | — | — | CTC1 | CS12 | CS11 | CS10 |

Рис. 3.12. Регистр TCCR1B таймера/счетчика Т/С1

Как уже было сказано ранее, разряды 0-2 используются для выбора частоты тактирования Т/С1 (см. табл. 3.5). Если разряд CTC1 установлен В лог. 1, то Т/С1 возвращается в состояние OxOOOO по импульсу такта системной синхронизации, следующего после совпадения с0держимого счетчика и регистра сравнения А. При работе в режиме ШИМ этот разряд на процесс работы никак не влияет.

Разряд ICESl определяет, каким образом должна осуществляться передача состояния счетчика В регистр захвата ICRl: no нарастающему (ICESI=1) или по ниспадающему фронту (ICES 1:0).

Разряд ICNCI определяет, должно ли быть активизировано подавление помех (если ICNCI = 0, то подавление помех отключено). Для подавления кратковременных импульсов помех, которые могут привести к ошибочному запуску, входной сигнал зондируется на протяжении четырех периодов такта системной синхронизации. Только после того как будут распознаны четыре последовательных низких или высоких уровня входного сигнала, что определяется разрядом ICESI, при активном подавлении помех будет выполнена запись текущего состояния счетчика В регистр ICRI.

## T/C2

Таймер/счетчик Т/С2 обычно имеет разрядность 8 бит и реализует функции сравнения на выходе и ШИМ, аналогичные Т/С1. Основная особенность Т/С2 заключается в том, что В качестве источника тактовых импульсов он может использовать генератор, независимый от системного. Для

управления T/C2 используются два регистра: ASSR (рис. 3.13) и TCCR2 (рис. 3.14).

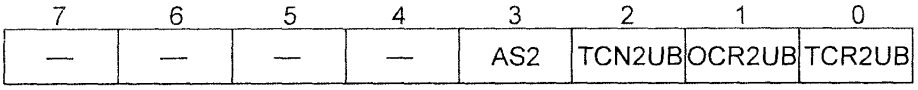



Рис. 3.13. Регистр ASSR таймера/счетчика T/C2

Если установить в лог. 1 разряд А32, то в качестве источника тактовых импульсов можно использовать внешний осциллятор. Оставшиеся три разряда (0-2) используются в программах для проверки того, что данные не записываются в регистры T/C2 В тот момент, когда они обновляются аппаратно. Такая проверка необходима по той причине, что осциллятор T/C2 работает асинхронно по отношению к системному осциллятору.

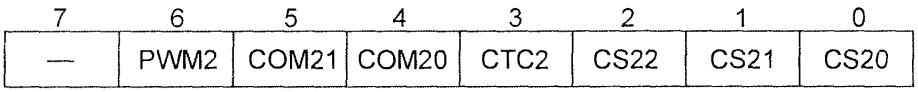



Рис. 3.14. Регистр TCCR2 таймера/счетчика T/C2

Установка в лог. 1 разряда PWM2 переводит T/C2 В режим ШИМ. Назначение разрядов COM21 и COM20 идентично назначению разрядов COM1x1 и COM1x0 таймера/счетчика T/C] - выбор режима сравнения на выходе.

Разряд CTC2 определяет, должен ли счетчик сбрасываться в нуль при совпадении его содержимого с регистром сравнения. Разряды 0-2 определяют частоту тактового сигнала, полученного с помощью предварительного делителя частоты такта системной синхронизации.

**Таймеры/счетчики микроконтроллеров PIC**

Описанное выше применение таймеров/счетчиков микроконтроллеров AVR справедливо также и для таймеров микроконтроллеров PIC. Здесь используются аналогичные принципы измерения ширины и частоты импульсов, а также широтно-импульсной модуляции, режимов сравнения и захвата.

В микроконтроллерах PIC могут использоваться три таймера: TMRO, TMRI и TMRZ.

TMRO

TMRO (рис. 3.14) - это 8-разрядный таймер/счетчик. Таким образом, счет для него ограничен диапазоном 0-255. Его тактирование реализуется от внешнего источника или на основании такта системной синхронизации.

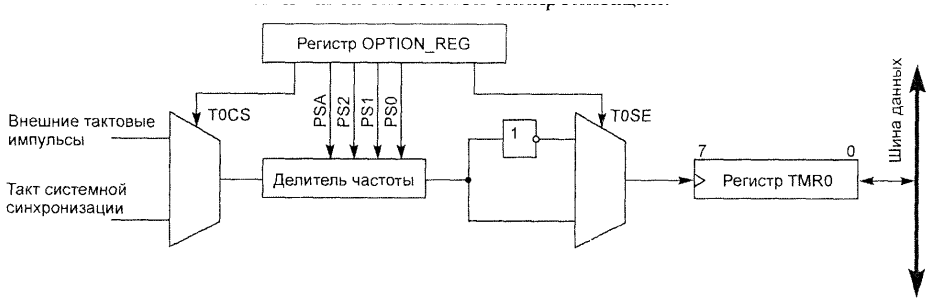



Рис. 3.14. Схема таймера/счетчика TMRO

Для управления работой таймера TMRO используются следующие разряды регистра OPTION_REG:

- разряд 5 - TOCS - определяет выбор источника синхроимпульсов (О - внутренний: 1 - внешний на входе ТОСК1);
- разряд 4 - TOSE - определяет выбор фронта, по которому происходит увеличение содержимого счетного регистра TMRO (О - по нарастающему; 1 - no ниспадающему фронту тактового сигнала);
- разряд 3 - PSA - использование предварительного делителя частоты (0 - делитель используется для управления таймером TMRO; 1 _- для управления сторожевым таймером);
- разряды 0-2 …- PSO, PSI, PS2 - выбор коэффициента деления частоты входного тактового сигнала (табл. 3.10).

Таблица 3.10. Назначение разрядов PS0-PS2 регистра OPTION_REG

| PS2 | PS1 | PS0 | Коэффициент деления частоты входного тактового сигнала |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 4 |
| 0 | 1 | 0 | 8 |
| 0 | 1 | 1 | 16 |
| 1 | 0 | 0 | 32 |
| 1 | 0 | 1 | 64 |
| 1 | 1 | 0 | 128 |
| 1 | 1 | 1 | 256 |

Для управления прерываниями от таймера TMRO используются следующие разряды регистра INTCON:

- разряд 2 - TOIF - флаг прерывания при переполнении TMRO;
- разряд 5 - TOIE - флаг разрешения прерывания при переполнении TMRO;
- разряд 7 ~- GIE - флаг общего разрешения прерываний.

**TMR1**

TMR1 (рис. 3.15) - это 16-разрядный таймер/счетчик, который может использоваться для формирования запросов на прерывание, подобно TMRO, или же работать в режимах захвата, сравнения и ШИМ.

Тактирование таймера TMR1 осуществляется от сигнала системной синхронизации или от специального генератора, предназначенного для работы с относительно медленными программными приложениями. Как правило, используется кварцевый резонатор частотой 32,768 кГц.

Для управления таймером TMR1 используется регистр T1CON (рис. 3.16). Назначение отдельных разрядов регистра TICON:

- TMRION - подключение таймера (0 - отключен, 1 - включен);
- TMRIC S - выбор источника тактирующих сигналов (0 - такт системной синхронизации; 1 - генератор 32,768 кГц);
- T1 SYNC - включение/отключение синхронизация специального генератора с генератором импульсов системной синхронизации (О - включена; 1 - отключена);
- TIOSC EN - разрешение/запрет тактирования таймера TMRI ОТ специального генератора (0 - генератор отключен; 1 - тактирование разрешено);
- T1CKPSO, T1CKPS1 - выбор коэффициента деления частоты (табл. 3.11).

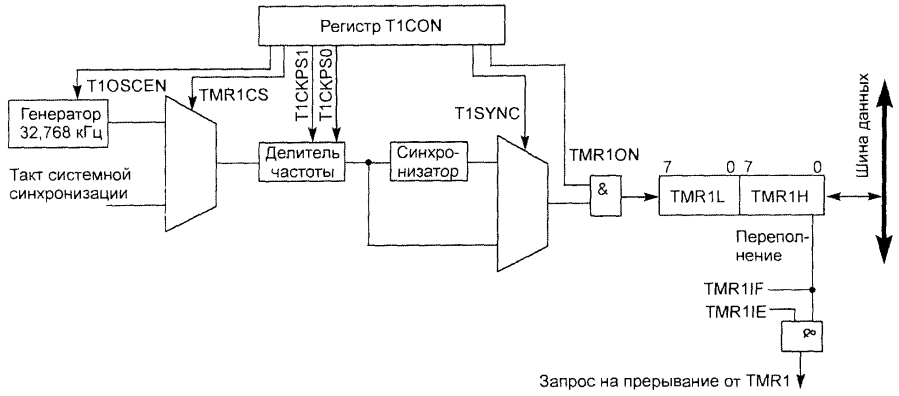



Рис. 3.15. Схема таймера/счетчика TMR1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | T1CKPS1 | T1CKPS0 | T1OSCEN | T1SYNC | TMR1CS | TMR1ON |

Рис. 3.16. Регистр T1CON микроконтроллеров PIC

Таблица 3.11. Назначение разрядов T1CKPS0-T1CKPS1 регистра T1CON

| T1CKPS1 | T1CKPS0 | Коэффициент деления частоты тактового сигнала |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 2 |
| 1 | 0 | 4 |
| 1 | 1 | 8 |

Счетный регистр таймера TMRI представляет собой регистровую пару TMRIH, TMRIL, а управление прерываниями осуществляется с помощью разрядов регистров PIRl и PIE]:

- регистр PIRI:
- разряд 0 - TMRIIF - флаг переполнения TMRI;
- разряд 2 - CCP1 IF - флаг прерывания при возникновении захвата по входу;

- регистр P1E1:
- разряд О – TMR1E - флаг разрешения прерывания при переполнении TMRI;
- разряд 2 - CCPHE - флаг разрешения прерывания при возникновении захвата по входу.

TMR2

Назначение таймера TMR2 (рис. 3.17) - измерение временных интервалов для реализации ШИМ, обеспечения определенной скорости обмена по последовательному порту и т.п. В этом смысле он псдобен таймеру TMRO.

Таймер TMR2 тактируется импульсами, следующими с частотой такта системной синхронизации, деленной на четыре. Каждый раз, когда содержимое счетного регистра TMR2 совпадает с содержимым регистра PR2, таймер автоматически сбрасывается в исходное (нулевое) состояние.

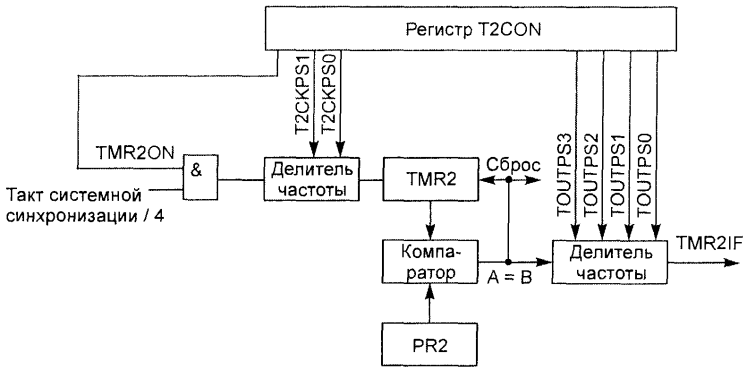



Рис. 3.17. Схема таймера/счетчика TMR2

При каждом совпадении TMR2 и PR2 генерируется запрос на прерывание, частоту возникновения которого можно также масштабировать с помощью выходного делителя частоты. Для управления таймером TMR2 используется регистр T2CON (рис. 3.18).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |

Рис. 3.18. Регистр TZCON микроконтроллеров PIC

Назначение отдельных разрядов регистра T2CON:

- T2CKPSO~T2CKPSI - управление предварительным делителем частоты (табл. 3.12);
- TMR2ON - подключение таймера (О - отключен, 1 - включен);
- TOUTPSO-TOUTPS3 -~- выбор коэффициента деления частоты запросов на прерывание при TMR2=PR2 (табл. 3.13).

Таблица 3.12. Назначение разрядов T2CKPS0-T2CKPS1 регистра T2CON

| T1CKPS1 | T1CKPS0 | Коэффициент деления частоты тактового сигнала |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 4 |
| 1 | x | 16 |

Таблица 3.13. Назначение разрядов TOUTPS0-TOUTPS3 регистра T2CON

| TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | Коэффициент деления частоты запросов |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 1 | 0 | 3 |
| 0 | 0 | 1 | 1 | 4 |
| 0 | 1 | 0 | 0 | 5 |
| 0 | 1 | 0 | 1 | 6 |
| 0 | 1 | 1 | 0 | 7 |
| 0 | 1 | 1 | 1 | 8 |
| 1 | 0 | 0 | 0 | 9 |
| 1 | 0 | 0 | 1 | 10 |
| 1 | 0 | 1 | 0 | 11 |
| 1 | 0 | 1 | 1 | 12 |
| 1 | 1 | 0 | 0 | 13 |
| 1 | 1 | 0 | 1 | 14 |
| 1 | 1 | 1 | 0 | 15 |
| 1 | 1 | 1 | 1 | 16 |

Для организации прерываний используются разряды TMR21E (флаг разрешения) и TMRZIF (флаг прерывания) регистров P1E1 и PIR1 соответственно.

Для того чтобы эффективно использовать таймер TMR2, используются следующие формулы:

$$T = (4 \cdot K1 \cdot K2 \cdot PR2) / F;$$

$$PR2 = T \cdot F / (4 \cdot K1 \cdot K2),$$

где T - требуемая временная задержка, K1 - коэффициент деления предварительного делителя частоты; K2 - коэффициент деления делителя частоты запросов на прерывание; PR2 - содержимое регистра PR2; F - частота системной синхронизации.

**Модуль CCP**

Таймеры TMR1 и TMR2 микроконтроллеров PIC применяются в составе модуля сравнения/захвата/ШИМ - CCP (Compare-Capture-PWM). Таких модулей может быть два: CCP1 и CCP2, - управление которыми реализовано с помощью регистров CCPXCON (рис. 3.19).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | DC1BX1 | DC1BX0 | CCP1M3 | CCP1M2 | CCP1M1 | CCP1M0 |

Рис. 3.19. Регистр CCPxCON микроконтроллеров PIC

Назначение отдельных разрядов регистра CCPXCON:

- I CCP1M0 - CCP1M3 _- выбор режима захвата/сравнения (табл. 3.14);
- DC 1 BXO - DC 1 BX1 - два младших разряда 10-разряцной ШИМ.

Таблица 3.14. Назначение разрядов CCP1M1-CCP1M3 регистра CCPxCON

| CCP1M3 | CCP1M2 | CCP1M1 | CCP1M0 | Значение |
|---|---|---|---|---|
| 0 | 0 | x | x | Модуль CCP отключен |
| 0 | 1 | 0 | 0 | Захват по каждому ниспадающему фронту |
| 0 | 1 | 0 | 1 | Захват по каждому нарастающему фронту |
| 0 | 1 | 1 | 0 | Захват по каждому 4-му нарастающему фронту |
| 0 | 1 | 1 | 1 | Захват по каждому 16-му нарастающему фронту |
| 1 | 0 | 0 | 0 | В случае совпадения на выходе — высокий уровень |
| 1 | 0 | 0 | 1 | В случае совпадения на выходе — низкий уровень |
| 1 | 0 | 1 | 0 | В случае совпадения — запрос на прерывание |
| 1 | 0 | 1 | 1 | Особый случай режима сравнения |
| 1 | 1 | x | x | Режим ШИМ |

**Режим ШИМ**

В режиме захвата (то есть, фиксации значения таймера в момент появления определенного условия) используются регистры CCPRIH, CCPRlL (В случае TMRI) или CCPR2H, CCPRL (В случае TMR2). В таком режиме таймер выполняет функции счетчика тактовых импульсов, и при наступлении условия захвата его содержимое переписывается в регистровую пару CCPRx.

В режиме сравнения модуль CCP формирует сигнал на выходе CCPx в том случае, когда содержимое счетного регистра становится равным значению, записанному в регистровой паре CCPRxL, CCPRXH. Этот режим обычно используется для выдачи сигналов на внешние устройства по истечении некоторого временного интервала.

В режиме ШИМ таймер работает как делитель частоты, формирующий период ШИМ-сигнала. Его значение постоянно сравнивается с содержимым регистра PR2, и при совпадении компаратор сбрасывает таймер в исходное состояние, после чего цикл повторяется. Параллельно организован контур сравнения, включающий в себя таймер, второй компаратор и регистр CCPRXH. Выходы обоих компараторов управляют триггером, выход которого соединен с выводом CCPx.

Вначале триггер устанавливается в “1” по сигналу сброса таймера, а по сигналу компаратора контура сравнения ~- сбрасывается в “0”. Таким образом, на выходе триггера формируется сигнал с периодом, определяемым содержимым регистров PR2 и CCPRXH.

**Сторожевой таймер**

Сторожевой таймер (watéhdog timer) _- встроенный таймер, тактируемый внутренним КС-осциллятором, который автоматически сбрасывает микроконтроллер при переполнении своего счетного регистра. В частности, он используется для предотвращения перехода микроконтроллера в режим

бесконечного цикла, когда на него невозможно повлиять извне. Обобщенная структурная схема сторожевого таймера показана на рис. 3.20.

В микроконтроллерах AVR и PIC управление сторожевым таймером несколько отличается. Так, в микроконтроллерах AVR для этого используется регистр управления WDTCR (адрес в области ввода/вывода - ox21, адрес SRAM - 0x41) (рис. 3.21).

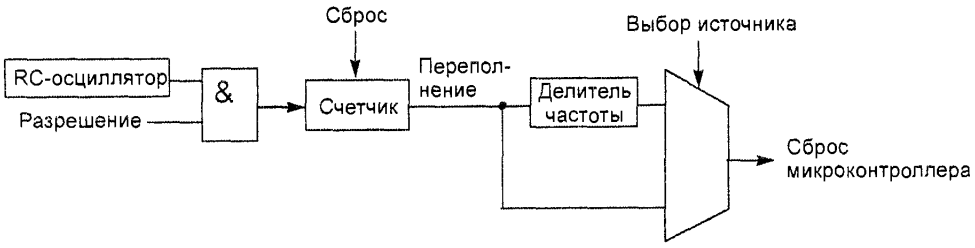



Рис. 3.20. Структурная схема сторожевого таймера

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | WDTOE | WDE | WDP2 | WDP1 | WDP0 |

Рис. 3.21. Регистр WDTCR микроконтроллеров AVR

Назначение отдельных разрядов регистра WDTCR:

- WDPO-WDP2 - выбор коэффициента деления частоты следования сигналов сброса (при этом период до наступления сброса зависит от рабочего напряжения процессора - табл. 3.15);
- WDE - включение/отключение сторожевого таймера (1 - включен);
- WDTOE - если сторожевой таймер должен быть отключен, следует установить этот разряд в лог. 1. После установки этого разряда он в течение четырех периодов такта системной синхронизации остается в состоянии лог.1, а затем аппаратно сбрасывается в лог. 0. Программа пользователя имеет возможность отключить сторожевой таймер посредством записи лог. 0 в разряд WDE только во время этих четырех тактов системной синхронизации.

Таблица 3.15. Назначение разрядов WDP0-WDP2 регистра WDTCR

| WDP2 | WDP1 | WDP0 | Коэффициент деления | Период до сброса (при Vcc = 5 В) | Период до сброса (при Vcc = 3 В) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 16 мс | 47 мс |
| 0 | 0 | 1 | 2 | 32 мс | 94 мс |
| 0 | 1 | 0 | 4 | 64 мс | 190 мс |
| 0 | 1 | 1 | 8 | 128 мс | 380 мс |
| 1 | 0 | 0 | 16 | 256 мс | 750 мс |
| 1 | 0 | 1 | 32 | 512 мс | 1,5 с |
| 1 | 1 | 0 | 64 | 1 с | 3 с |
| 1 | 1 | 1 | 128 | 2,1 с | 6 с |

В системе команд AVR сторожевой таймер сбрасывается в исходное состояние по команде wdr.

В микроконтроллерах PIC для управления сторожевым таймером предназначен рассмотренный выше регистр OPTION. Для этого разряд PSA должен быть установлен в лог. 1, чтобы предварительный делитель частоты был переключен на использование совместно со сторожевым таймером, а не с TMRO. Коэффициент деления выбирается с помощью разрядов PS2-PSO (табл. 3.16).

В отличие от микроконтроллеров AVR, В микроконтроллерах PIC отсутствует возможность включать/отключать сторожевой таймер с помощью регистра управ ления. Единственный способ предотвратить сброс от сторожевого таймера- периодически выполнять ассемблерную команду clrwdt.

Таблица 3.16. Выбор коэффициента деления частоты следования сигналов сброса от сторожевого таймера в микроконтроллерах PIC

| PS2 | PS1 | PS0 | Коэффициент деления | Период до сброса |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 18 мс |
| 0 | 0 | 1 | 2 | 36 мс |
| 0 | 1 | 0 | 4 | 72 мс |
| 0 | 1 | 1 | 8 | 144 мс |
| 1 | 0 | 0 | 16 | 288 мс |
| 1 | 0 | 1 | 32 | 576 мс |
| 1 | 1 | 0 | 64 | 1,2 с |
| 1 | 1 | 1 | 128 | 2,3 с |

**Параллельные порты ввода/вывода**

Параллельные порт - это особые устройства ввода/вывода, позволяющие передавать во внешний мир или принимать Одновременно восемь разрядов данных. Для обозначения портов используются латинские буквы А, В, С и т.д. Количество портов ввода/вывода варьируется В зависимости от модели микроконтроллера.

В микроконтроллерах AVR каждому параллельному порту ввода/вывода поставлены в соответствие три регистра (букве х соответствует имя порта А, В и т.д.):

- DDRX - регистр направления передачи данных - определяет, является тот или иной вывод порта входам или вых0дом; если некоторый разряд регистра DDRx содержит лог. 0, то соответствующий ВЫВОД порта сконфигурирован как ВХОД, В противном случае - как ВЫХОД;
- РОВТХ - регистр порта - если ВЫВОД выполняет роль выхода, то в соответствующий разряд записывается значение, предназначенное для вывода; если вывод выполняет роль входа, то лог. 0 в некотором разряде регистра PORTx соответствует высокоомный ВХОД, а лог. 1 - ВХОД, нагруженный подтягивающим сопротивлением;
- PIN х - регистр ВЫВОДов порта - в отличие от регистров DDRx и PORTx доступен только Для чтения и позволяет считать входные данные порта на внутреннюю шину микроконтроллера.

Выводы портов зачастую выполняют различные альтернативные функции при работе со внутренними и периферийными модулями микроконтроллеров AVR. Так, к примеру, в некоторых моделях в качестве внешних тактовых входов таймеров/счетчиков T/CO и T/C1 используются разряды 0 и 1 порта B или 4 и 5 порта D. Точное назначение выводов портов следует сверять по спецификации микроконтроллера.

В микроконтроллерах PIC каждому параллельному порту ввода/вывода поставлены в соответствие два регистра:

- I PORTX - регистр Данных порта;
- TRISx …… регистр направления передачи Данных через выводы порта

l (лог. 1 в некотором разряде этого регистра соответствует режим ввода в лог. 0 -~ режим вывода).

**Режим PSP порта D микроконтроллеров PIC**

В микроконтроллерах PIC серии 18Cx порт D может работать в режиме управляемого параллельного порта PSP (Parallel Slave Port). Это означает, что он действует как регистр. который может быть подключен к шине Другого микроконтроллера, обмениваясь с ним данными. В режиме PSP, как и в случае обмена данными с любым периферийным устройством. используются сигналы RD (чтение), WR (запись) и CS (выбор кристалла) - разряды O_2 порта E (пример - рис. 3.22).

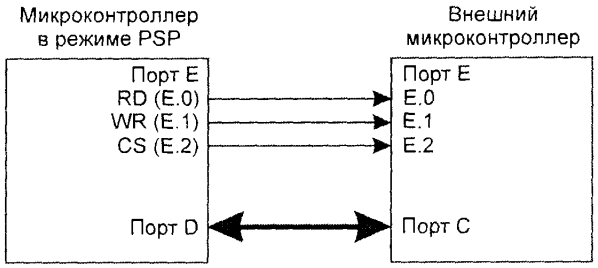



Рис. 3.22. Пример подключения внешнего микроконтроллера PIC B режиме РЭР

Для управления режимом PSP используется регистр TRISE (рис. 3.23).

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IBF | OBF | IBOV | PSPMODE | — | TRISE2 | TRISE1 | TRISE0 |

Рис. 3.23. Регистр TRISE микроконтроллеров PIC

Режим PSP активизируется путем установки в лог. l разряда PSPMODE. Прерывания разрешаются установкой в лог. 1 разряда PSPIE (разряд 7) регистра PIE1, а запросы формируются в разряде PSPIF (разряд 7) регистра PlR1. C помощью разрядов 0-2 регистра TRISE осуществляется выбор режима для соответствующих разрядов порта E.

Когда на линиях CS и RD (выводы REZ и REO) одновременно появляется низкий уровень сигнала, содержимое регистра OUTREG ВЫВОДИТСЯ через порт D. При записи в регистр OUTREG устанавливается в лог. 1 разряд OBF регистра TRISE это означает, что выходной буфер заполнен данными. После передачи данных разряд OBF автоматически сбрасывается в лог. 0.

Когда на линиях CS и WR (выв0ды R132 и REl) одновременно появляется низкий уровень сигнала, осуществляется прием данных через порт D. Принятая величина сохраняется в регистре INREG, при этом автоматически устанавливается в лог. 1 разряд IBF регистра TRISE. После программного считывания содержимого регистра INREG этот разряд автоматически сбрасывается в лог. 0.

Если ранее принятый байт не считывается до поступления следующего байта в регистр INREG, устанавливается в лог. 1 разряд IBOV регистра TRISE, указывающий на переполнение входного буфера.

## 3.2 **Язык С и директивы процессора**

В этой главе кратко рассматривается синтаксис языка С, а также директивы препроцессора в том объеме, которого будет достаточно для изучения примеров из следующих двух глав. Даже если читатель уже знаком с представленными здесь вопросами, все же рекомендуем просмотреть эту главу, чтобы выяснить особенности, характерные для компиляторов WinAVR и CCS-PICC.

**Вводные понятия**

Прежде всего, рассмотрим такие вводные понятия как комментарии, ключевые слова. идентификаторы, литералы, операторы и знаки пунктуации.

Комментарий - это некоторый поясняющий текст, который при компиляции не учитывается. Комментарии бывают многострочными (начинаются с символов и заканчиваются символами * / ) и однострочными (начинаются 0 символов). В последнем случае комментарием считается вся часть строки, расположенная справа от символов / / . Примеры:

```
/* Многострочный комментарий часто размещают в начале файла,
   где он содержит имя автора и описание программы */
#include <avr/io.h> //Подключаем заголовочный файл io.h
```

Идентификатор - это последовательность букв, цифр и символов подчеркивания “____”, которая не должна начинаться 0 цифры, используемая для именования различных программных элементов, наподобие переменных, констант, функций, типов и т.д. Регистр букв имеет значение.

Ключевое слово - это зарезервированное слово четко определенного назначения. Ключевые слова не могут использоваться в качестве идентификаторов:

```
asm, auto;
bit, bool, break;
case, char, const, continue;
default, defined, do, double;
else, enum, explicit, extern;
false, float, for;
goto; if, inline, int;
long;
register, return;
short, signed, sizeof, static, struct, switch;
true, typedef;
union, unsigned;
void; while.
```

asm,auto;

bit,bool,break;

case,char,const,continue;

Литерал - постоянное значение некоторого типа, используемое в выражениях. Примеры числовых литералов:

10 — число 10 в десятичной форме;
0xA — число 10 в шестнадцатеричной форме (префикс 0x);
0b1010 — число 10 в двоичной форме (префикс 0b);
012 — число 10 в восьмеричной форме (префикс 0);
10.5 — число с плавающей точкой;
105e-1 — число 10,5 в экспоненциальной форме;
10U — беззнаковая константа (суффикс U);
10L — знаковая константа (суффикс L).

Символьные литералы заключаются в одинарные кавычки, например, 'В' для обозначения непечатаемых и специальных символов В литералах используются так называемые езсаре-последовательности:

'\a' — звуковой сигнал;
'\b' — клавиша <Backspace>;
'\f' — прогон листа;
'\n' — символ перевода строки;
'\r' — возврат каретки;
'\t' — горизонтальная табуляция;
'\v' — вертикальная табуляция;
'\0' — нулевой символ;
'\\' — обратная косая;
'\'' — апостроф.

Кроме того, любой символ можно представить с помощью литерала по его АЗСП-коду, например, литерал ' \t ' равнозначен ' \x09' (в шестнадцатеричном представлении).

Строковые литералы ограничиваются двойными кавычками, а В памяти хранятся как последовательности символов, заканчивающиеся нулевым символом О ‘ . Специальные символы внутри строки должны предваряться обратной косой

Примеры строковых литералов:

"" — пустая строка: один символ '\0';
"В" — два символа: 'В' и '\0';
"A\tB\n" — пять символов: 'A', табуляция, 'B', перевод строки, '\0'.

Оператор - это символ, указывающий компилятору, какие действия выполнить над операндами. Некоторые символы могут трактоваться по-разному в зависимости от контекста. Например, знак “-” может использоваться для изменения знака числа или в качестве оператора вычитания. Операторы, соединяющие операнды, представляют собой выражения. Выражения могут быть заключены в круглые скобки и отделяются друг от друга символом точки с запятой. Приоритетность выполнения операторов в выражениях языка С указана в табл. 3.16 (приоритет с меньшим номером уровня - выше).

Таблица 3.16. Приоритетность выполнения операторов в выражениях языка С

| Уровень | Операторы | Категория | Описание |
|---|---|---|---|
| 1 | () | | Круглые скобки |
| | [] | | Элемент массива |
| | . | Доступ к данным | Обращение к элементу структуры, например: PORTB.1 — разряд 1 порта В |
| | -> | | Обращение к элементу структуры, определенной указателем, например: pStruct->x — элемент x структуры, на которую указывает pStruct |
| | ++, -- (постфиксы) | Арифметические | Операторы автоинкремента и автодекремента после того как выражение, в котором задействованы соответствующие операнды, вычислено. Примеры: a = b++; равнозначно a = b; b = b+1; a = b--; равнозначно a = b; b = b-1; |
| 2 | ++, -- (префиксы) | | Операторы автоинкремента и автодекремента перед тем как выражение, в котором задействованы соответствующие операнды, будет вычислено. Примеры: a = ++b; равнозначно b = b+1; a = b; a = --b; равнозначно b = b-1; a = b; |
| | ! | Логические | Логическое (унарное) отрицание |
| | ~ | Поразрядные | Поразрядное отрицание |
| | & | Доступ к данным | Адрес |
| 3 | +, - (унарные) | Арифметические | Изменение знака операнда |
| | * (унарный) | Доступ к данным | Разыменование указателя |
| 4 | * (бинарный) | Арифметические | Умножение |
| | / | | Деление |
| | % | | Остаток от деления |
| 5 | +, - (бинарные) | | Сложение и вычитание |
| 6 | << | Поразрядные | Поразрядный сдвиг влево |
| | >> | | Поразрядный сдвиг вправо |
| 7 | <, >, <=, >= | Сравнения | Меньше, больше и т.д. |
| 8 | ==, != | | Равно, не равно |
| 9 | & | Поразрядные | Поразрядное "И" |
| 10 | ^ | Поразрядные | Поразрядное "Исключающее ИЛИ" |
| 11 | \| | Поразрядные | Поразрядное "ИЛИ" |
| 12 | && | Логические | Логическое "И" |
| 13 | \|\| | Логические | Логическое "ИЛИ" |
| 14 | = | Присваивание | Возможны также сочетания оператора присваивания с арифметическими и поразрядными операторами, например: a += b; равнозначно a = a + b; a *= b+c; равнозначно a = a * (b + c); |

Объединенные по некоторому признаку последовательности выражений заключаются в фигурные скобки { }. Так, к примеру, обозначаются границы функций, а также блоки выражений в циклических и условных конструкциях (см. ниже соответствующие разделы этой главы).

## Структура программы на С

Структуру программы, написанной на языке С, изучим на основании примера SOS.c (СМ. листинг).

```
//Директивы препроцессора
#include <18F458.h>
#use delay(clock=20000000)
#fuses HS,WDT

//Объявления глобальных типов, переменных и констант
...

//Функции
Function1
{
  //Объявления локальных типов, переменных и констант
  ...
  //Операторы
  ...
}
...
FunctionN
{
  //Объявления локальных типов, переменных и констант
  ...
  //Операторы
  ...
}

//Главная функция программы
int main (void)
{
  //Объявления локальных типов, переменных и констант
  ...
  //Операторы
  ...
}
```

Программы обычно начинаются с директив препроцессора (начинаются 0 символа “#”), которые, по сути, не являются конструкциями языка С и обрабатываются до фактической компиляции программы. Их смысл - подстановка некоторого кода в программу. Так, к примеру, очень часто используется директива include, которая включает в файл с исходным кодом программы текст внешнего заголовочного файла (с расширением .h). Заголовочные файлы содержат определения глобальных типов, констант, переменных и функций.

## Типы данных, переменные, константы

Тип данных определяет диапазон допустимых значений и пространство, отводимое в памяти данных, для переменных, констант и результатов, возвращаемых функциями. В различных компиляторах с языка С могут использоваться собственные типы данных, однако все они базируются на стандартных типах, перечисленных в табл. 3.17.

Таблица 3.17. Стандартные типы данных языка С

| Тип | Название | Размер, в битах | Диапазон значений |
|---|---|---|---|
| bit | Бит | 1 | 0, 1 |
| char | Символ | 8 | –128..127 |
| int | Целое число | 16 | –32768..32767 |
| float | Вещественное число | 32 | $\pm1{,}175\cdot10^{-38}$.. $\pm3{,}402\cdot10^{38}$ |
| long int | Длинное целое | 32 | –2147483648..2147483647 |
| unsigned char | Беззнаковый символ или логическое значение | 8 | 0..255<br>TRUE, FALSE |
| unsigned int | Беззнаковое целое | 16 | 0..65535 |
| unsigned long int | Беззнаковое длинное целое | 32 | 0..4294967295 |

Рассмотрим некоторые определения подробнее, чтобы понять, каким образом получаются перечисленные в табл. 3.17 диапазоны допустимых значений.

Бит - это базовая единица информации в вычислительной технике, означающая одно из двух состояний: 0 или 1, высокий уровень сигнала или низкий уровень сигнала. К примеру, при подключении светодиодов к выводам микроконтроллера для каждого такого вывода программист, опять таки, оперирует понятием бита, поскольку в данном случае речь идет об двух уровнях напряжения, один из которых включает, а другой - отключает светодиод. В общем случае, говорят, что бит содержит логический “0” или логическую “1”. Следующая фундаментальная единицы информации - байт, состоящий из восьми битов. С его помощью можно представить 256 различных состояний (0..255), что иллюстрирует табл. 3.18.

Таблица 3.18. Значения, которые можно получить с помощью одного байта

| Значение | Двоичное представление | Восьмеричное представление | Шестнадцатеричное представление |
|---|---|---|---|
| 0 | 0b00000000 | 0000 | 0x00 |
| 1 | 0b00000001 | 0001 | 0x01 |
| 2 | 0b00000010 | 0002 | 0x02 |
| 3 | 0b00000011 | 0003 | 0x03 |
| 4 | 0b00000100 | 0004 | 0x04 |
| 5 | 0b00000101 | 0005 | 0x05 |
| 6 | 0b00000110 | 0006 | 0x06 |
| 7 | 0b00000111 | 0007 | 0x07 |
| 8 | 0b00001000 | 0010 | 0x08 |
| 9 | 0b00001001 | 0011 | 0x09 |
| 10 | 0b00001010 | 0012 | 0x0A |
| 11 | 0b00001011 | 0013 | 0x0B |
| 12 | 0b00001100 | 0014 | 0x0C |
| 13 | 0b00001101 | 0015 | 0x0D |
| 14 | 0b00001110 | 0016 | 0x0E |
| 15 | 0b00001111 | 0017 | 0x0F |
| 16 | 0b00010000 | 0020 | 0x10 |
| ... | ... | ... | ... |
| 254 | 0b11111110 | 0376 | 0xFE |
| 255 | 0b11111111 | 0377 | 0xFF |

**Правила преобразований из одной системы счисления в другую**

Схема преобразования из некоторой системы счисления в десятичную очень проста: каждый разряд умножаем на основание системы, возведенное в соответствующую разряду степень (начиная с О), и затем складываем полученные произведения для всех разрядов. Пример для десятичного числа 100:

$0b01100100 = 0\cdot2^7 + 1\cdot2^6 + 1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 =$
$0 + 64 + 32 + 0 + 0 + 4 + 0 + 0 = 100;$

$0144 = 1\cdot8^2 + 4\cdot8^1 + 4\cdot8^0 = 64 + 32 + 4 = 100;$

$0x64 = 6\cdot16^1 + 4\cdot16^0 = 96 + 4 = 100.$

Для преобразования из десятичного представления в другую систему счисления число следует разделить на основание системы, запомнить остаток,

затем частное еще раз разделить на основание системы, опять запомнить остаток и т.д. до тех пор, пока не будет получено неделимое частное. Это частное является старшим разрядом полученного представления, а остальные разряды формируются из остатков, начиная от последнего к первому.

Пример преобразования числа 100 в двоичную систему счисления:
100 / 2 = 50, остаток 0
50 / 2 = 25, остаток 0
25 / 2 = 12, остаток 1
12 / 2 = 6, остаток 0
6 / 2 = 3, остаток 0
3 / 2 = 1, остаток 1
1 - на 2 не делится. Результат: 100 = ОЬ1100100.

Пример преобразования числа 100 в восьмеричную систему счисления:
100 / 8 = 12, остаток 4
12 / 8 = 1, остаток 4
1 - На 8 не делится. Результат: 100 = 0144.

Пример преобразования числа 100 в шестнадцатеричную систему счисления:
100/ 16 = 6, остаток 4
6 - на 16 не делится. Результат: 100 = ox64.

**Тип char**

Байту данных в языке С соответствует тип char. ОН получил свое название от английского слова “character”, поскольку чаще всего используется в программах для хранения символов (то есть, их АЗСП-кодов). Тем не менее, 0 его помощью можно обращаться и к небольшим целым числам в диапазоне от -128 до +127, а для типа unsigned char - от 0 до 255, поскольку в этом случае знаковый разряд не используется (использование слова unsigned имеет такой же смысл и в случае целочисленного типа int).

**Пользовательские типы**

Язык С позволяет, кроме стандартных, объявлять собственные типы. Для этого используется ключевое слово typede f, например:

```
typedef unsigned char byte; //объявляем тип byte
typedef unsigned int word;  //объявляем тип word
```

Другими словами, для объявления пользовательского типа используется конструкция вида

```
typedef стандартный_тип идентификатор_пользовательского_типа;
```

Так, в компиляторе WinAVR определены следующие пользовательские типы:

typedef signed char **int8_t** — в заголовочном файле `inttypes.h`;

typedef unsigned char **uint8_t** — в заголовочном файле `inttypes.h`;

typedef int **int16_t** — в заголовочном файле `inttypes.h`;

typedef unsigned int **uint16_t** — в заголовочном файле `inttypes.h`;

typedef long **int32_t** — в заголовочном файле `inttypes.h`;

typedef unsigned long **uint32_t** — в заголовочном файле `inttypes.h`;

typedef long long **int64_t** — в заголовочном файле `inttypes.h`;

typedef unsigned long long **uint64_t** — в файле `inttypes.h`;

typedef struct {int quot; int rem;} **div_t;** — в заголовочном файле `stdlib.h` (этот тип используется стандартной функцией `div()`);

typedef struct {long quot; long rem;} **ldiv_t;** — в заголовочном файле `stdlib.h` (этот тип используется стандартной функцией `ldiv()`).

В компиляторе CCS-PICC также определены некоторые пользовательские целочисленные типы:

I int 1, short – определяют тип размером в один бит;

int8 - аналог стандартного типа char;

int 1 6 - аналог стандартного типа int;

int32 - аналог стандартного типа long int.

**Переменные**

Переменная - это именованная величина определенного типа, которая может изменяться в ходе выполнения программы. Для объявления переменных (т.е., выделения для них памяти) в программе на С используется следующая конструкция:

Тип переменной Идентификатор 1, идентификатор2,

Например:

int i; //Объявление целочисленной переменной i

char cl, c2; //Объявление символьных переменных c1 и c2

Для доступа к переменной в программе используется соответствующий Идентификатор (обязательно после объявления переменной). Значения, присваиваемые переменной, должны соответствовать ей по типу (или правилам приведения типов, рассматриваемым ниже). Примеры:

```
i = 2;     //Ошибка! Переменная I еще не объявлена
int i;     //Объявление целочисленной переменной i
float f;   //Объявление вещественной переменной f
i = 2;     //Переменной i присвоено значение 2
f = 3.3    //Переменной f присвоено значение 3,3
f = i;     //Переменной f присвоено значение переменной i
           //(в данном случае будет выполнено автоматическое
           //приведение типов, т.е. f = 2.0)
```

По области видимости переменные могут быть глобальными и локальными.

К глобальным переменным имеют доступ все функции программы. Такие переменные объявляются в программе перед объявлением всех функций. К локальным переменным имеет доступ только та функция, в которой они объявлены.

Функции рассматриваются ниже в соответствующем разделе главы.

**Область видимости переменных**

Имена переменных обладают определенной областью видимости, которая подразумевает, что компилятор использует переменные в соответствии с тем, где они находятся. Имена переменных, объявленных внутри функции, имеют область видимости, ограниченную конкретной функцией. Например, в нескольких функциях можно объявить переменную int i, которая в каждой функции не будет иметь никакой связи с аналогичными переменными В других функциях. Точно так же и переменная, объявленная внутри блока (ограничен фигурными скобками остается локальной по отношению к этому блоку.

Глобальные переменные имеют область видимости. которая начинается от места их объявления и продолжается до конца программного файла. Для того чтобы глобальную переменную можно было использовать в Других файлах, ее нужно объявить с помощью ключевого слова extern:

extern int n;

Объявленную таким образом переменную. прежде, чем ее использовать, следует обязательно инициализировать во внешнем файле некоторым значением.

**Константы**

Константа - это именованная величина определенного типа, которая, в отличие от переменной, не может изменяться в ходе выполнения программы, а имеет конкретное значение` определенное в момент объявления. Для объявления констант в программе на С используется следующая конструкция:

const тип_константы идентификатор : значение;

Например:

const int i = 10; //Объявление целочисленной константы i

Величина, объявленная как константа, будет размещена компилятором в памяти программ, а не в ограниченной области переменных в RAM.

Для доступа к константе используется ее Идентификатор. Примеры:

```
c = 'A';        //Ошибка! Константа c еще не объявлена
int i;          //Объявление целочисленной переменной i
const c = 'A'; //Объявление константы c
i = 2;          //Переменной i присвоено значение 2
c = i;          //Ошибка! Попытка присвоить значение константе
i = c;          //Переменной i присвоено значение константы c.
      //В данном случае будет выполнено автоматическое
      //приведение типов, т.е. i = 65 (ASCII-код символа 'A')
```

**Перечислимые типы**

Перечислимый тип - это объявление списка целочисленных констант, которые можно явно не инициализировать (в этом случае компилятор считает,

что первая константа в списке принимает значение 0, вторая - 1 и т.д.). Для подобного объявления используется ключевое слово enum:

```
 int n;
enum (zero, one, two); //zero = O; one = 1; two = 2
n = one; //n = 1
```

Если требуется изменить начальное значение для списка констант, то можно указать его явно при объявлении, например:

```
enum (three = 3, four, five); //three = 3; four = 4; five = 5
```

**Приведение типов**

Приведение типов - это принудительное преобразование значения одного типа к другому, совместимому с исходным. Это важно при выполнении арифметических операций, когда полученные значения могут выходить за допустимые пределы.

Приведение типов бывает явным и неявным. Неявное приведение типов используется в операторах присваивания, когда компилятор сам выполняет необходимые преобразования без участия программиста. Примеры подобного приведения уже рассматривались выше в подразделах, посвященных переменным и константам.

Для явного приведения типа некоторой переменной перед ней следует указать в круглых скобках имя нового типа, например:

```
int X;
int Y = 200;
char C = 30;
X = (int)C * 10 + Y; //Переменная C приведена к типу int
```

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение C * 10 - это восьмиразрядное умножение (разрядности типа char) и вместо корректного значения 300 (OleC) в стек было бы помещено урезанное значение 44 (Ox2C). Таким образом, в результате вычисления выражения C * 10 + Y переменной X было бы присвоено значение 64 O, а не корректное 3200. В результате приведения типа переменная C распознается компилятором как 16-тиразрЯДная, и описанной выше ошибки не возникает.

**Оператор sizeof**

Оператор sizeof применяется для вычисления размера области памяти (в байтах), отводимой под некоторую переменную, результат выражения или тип.

Например:

```
int a;
float f;
a = sizeof(int); //a = 2
```

a = sizeof(f); //a = 4
f = 3.3;
a = sizeof(f + a); //a = 4, поскольку тип результата - float

**Функции**

Функция представляет собой “контейнер”, в котором выполняется некоторый фрагмент программного кода. Использование функций упрощает написание и отладку программ` поскольку в них удобно размещать повторяющиеся группы операторов. Любая программа на языке С с0дер>кит главную функцию под названием main ( ) . Эта функция при запуске программы выполняется первой.

Пользовательские функции определяются после директив препроцессора и глобальных объявлений типов, переменных и констант в файле с исходным кодом или в заголовочном файле. При этом используется следующий синтаксис:

```
Тип_возвращаемого_значения Имя_функции(Список_параметров)
{
 //Тело функции
}
```

В качестве типа возвращаемого значения может использоваться ключевое слово void. Это означает, что функция или не возвращает никакого значения (в некоторых языка программирования такие функции называют процедурами).

Функцию вызывают по ее имени с указанием в круглых скобках перечня передаваемых параметров (если их нет, то в скобках ничего не указывается), например:

```
void Function1(int n, char c)
{
  ...
}

int Function2()
{
  ...
}

int main()
{
  int x;
  char y;
  Function1(x, y);
  x = Function2();
}
```

Параметры - это идентификаторы, которые могут использоваться внутри функции. Вместо них подставляются соответствующие значения, указанные при вызове функции (если в функцию передается более одного параметра, то они отделяются друг от друга занятыми как при объявлении, так и при вызове).

Значения, переданные в функцию, фактически не изменяются, а просто копируются в параметры, который в этом смысле выполняют роль локальных переменных. При этом следует следить за тем, чтобы тип передаваемых значений соответствовал типу параметров, объявленных в заголовке функции.

**Возвращаемые значения**

Если функция предназначена для возврата значения некоторого типа, то для этого в ее теле используют ключевое слово return, после которого (через пробел) указывают возвращаемое значение. При этом все операторы после слова return игнорируются, и происходит возврат в вызывающую функцию. Пример:

```
...
int power3(int n)
{
  return n*n*n;
}

void main()
{
  int x;
  x = power3(2); //x = 8
}
```

Слово return может также использоваться без указания возвращаемого выражения. В этом случае оно просто означает выход из функции.

**Прототипы функций**

В обычном варианте функции используются только после их определения, однако бывают случаи, когда функции вызывают друг друга, и организовать их “правильное” определение невозможно. Обойти подобную проблему позволяют прототипы функций, которые представляют собой объявление до определения.

Такое объявление представляет собой только заголовок функции, причем в списке параметров указывают только типы, без идентификаторов, например:

```
...
int f1(int);
void f2(int, int);

int f1(int x)
{
  …
}

void f2(int a, int b)
{
  ...
}
```

Прототипы функций часто используются в заголовочных файлах, включаемых в текст программы с помощью директивы препроцессора #include.

**Классы памяти при объявлении локальных переменных**

Локальные переменные могут быть объявлены внутри функций как принадлежащие к одному из трех классов памяти:

- auto (значение по умолчанию, можно явно не указывать) - при объявлении переменная не инициализируется никаким значением (значение - текущее содержимое области памяти, отведенной псд переменную); при выходе из функции переменная удаляется из памяти;
- static ... статическая переменная доступна только в пределах функции,

хотя память для нее вьяделяется в пространстве глобальных переменных; при первом обращении К функции инициализируется нулевым значением, и после ВЫХОДа из функции из памяти не удаляется (таким образом, при последующих обращениях К функции в ней содержится старое значение);

- register - аналог автоматической локальной переменной за тем исключением, что компилятор попытается выделить для нее не область памяти данных, а рабочий регистр микроконтроллера, что значительно ускоряет обращение к значению переменной.

Пример использования статической переменной:

```
...
int plus5()
{
  static int x;
  return x + 5;
}

void main()
{
  int y;
  y = plus5(); //y = 5
  y = plus5(); //y = 10
  y = plus5(); //y = 15
}
```

**Рекурсия**

Рекурсия - это вызов функцией самой себя. Эта возможность бывает трудна в понимании, и потому не удивительно, что эксперты по языку С так любят рекурсивные функции. Пример рекурсивного вызова:

```
void f1(int n)
{
   int x;
   f1(x);
}
```

Несмотря на сложность восприятия, рекурсия довольно часто используется в стандартных библиотечных функциях, а также во многих алгоритмах сортировки. Тем не менее, при программировании микроконтроллеров использование рекурсии, как правило, чревато проблемами из-за ограниченного объема оперативной памяти. Дело в том, что при каждом вызове рекурсивной функции часть памяти расходуется на сохранение данных, помещаемых в стек. Эти данные хранятся там до тех пор, пока не будет выполнен возврат из функции. Таким образом, когда рекурсивная функция снова и снова вызывает саму себя, в стеке остается все меньше и меньше свободной памяти.

Можно сказать, что в подавляющем большинстве случаев использование рекурсии при программировании микроконтроллеров - это надежный способ быстрого и непредсказуемого заполнения стека. Это очень часто приводит к возникновению ошибочного состояния, называемого переполнением стека. Область стека обычно размещается в верхней части памяти и растет “вниз”, тогда как область переменных размещается в нижней части памяти и растет “вверх”. Поэтому если объем данных, помещаемых в стек, превысит размер области

стека, эти данные могут достигнуть области переменных и затереть собой ее значения. При этом ошибку переполнения стека не всегда легко выявить, поскольку она может проявляться лишь периодически.

**Структуры**

Структура - это особый тип данных, состоящий из нескольких разнотипных переменных (полей). В общем случае объявление структуры имеет следующий вид:

```
struct имя_структуры {
   тип поле_1;
   ...
   тип поле_N;
};
```

Как и любой другой тип, структуру можно в дальнейшем использовать для объявления переменных, например:

```
struct MyStructure { //Объявление структуры MyStructure
   int Field1;
   char Field2;
   float Field3;
};
//Объявление переменных YourStruct и OurStruct типа
MyStructure
struct MyStructure YourStruct, OurStruct;
```

Допускается инициализация полей непосредственно при объявлении переменных-структур с помощью перечня значений в фигурных скобках, например:

```
struct DATE {
   int Day;
   int Month;
   int Year;
}
struct DATE MyBirthday = {7, 8, 1974};
```

Для доступа к полям структуры в программе используют запись вида имя_структуры. поле. То есть, в представленном выше примере структуры DATE для инициализации полей можно было воспользоваться следующими операторами:

```
MyBirthday.Day = 7;
MyBirthday.Month = 8;
MyBirthday.Year = 1974;
```

Структуры, в свою очередь, могут быть полями других структур, например:

```
struct ME {
   char MyName[30];  //Строка длиной 30 символов - имя
   struct DATE MyBirthday; //Структура, хранящая день рождения
}
...
struct ME MyData;
MyData.MyName = "John Smith";
MyData.MyBirthday.Day = 7;
MyData.MyBirthday.Month = 8;
MyData.MyBirthday.Year = 1974;
```

Структуры могут выступать в качестве параметров функций, а также возвращаемого результата. Ниже представлен пример функции, возвращающей структуру типа DATE:

```
struct DATE January1(int CurYear)
{
  struct DATE Jan01;
  Jan01.Day = 1;
  Jan01.Month = 1;
  Jan01.Year = CurYear;
  return Jan01;
}
...
struct DATE BeginOfTheYear;
BeginOfTheYear = January1(2006);
```

**Указатели и адреса переменных**

Указатель - это переменная, содержащая адрес некоторого элемента данных (переменной, константы, функции, структуры). В языке С указатели тесно связаны с обработкой массивов и строк, которые будут рассмотрены в следующем разделе.

Для объявления переменной как указателя используется оператор

```
int *p; //p – указатель на целое число
```

Для присвоения адреса некоторой переменной указателю используется оператор &, например:

```
char *p;  //указатель на символ
char c;  //символьная переменная
c = 'A';
p = &c; // p содержит адрес переменной c (указывает на 'A')
```

Для того чтобы извлечь значение переменной, на которую указывает указатель, используется оператор разыменования

```
char *p;
char c, b;
c = 'A';
p = &c;
b = *p;    //Теперь b = 'A'
```

Аналогичным образом, этот оператор можно использовать и для присвоения некоторого значения переменной, на которую указывает

```
char *p;
char c, b;
b = 'A';
p = &c;
*p = b;    //Теперь c = 'A'
```

Применительно к программированию микроконтроллеров, указатели можно использовать, к примеру, для записи Данных в порт ввода/вывода. Предположим, регистр данных порта расположен в памяти по адресу ox16. В этом случае, для записи в него значения OXFF можно воспользоваться следующим фрагментом программного кода:

```
unsigned char *thePort;
thePort = 0x16;
*thePort = 0xFF;
```

**Передача в функции параметров по ссылке**

С помощью указателей, используемых в качестве параметров функций, можно организовать возврат более Одного значения. Рассмотрим следующий пример:

```
int SumAndDiv(int *a, int *b)
{
  int bufA, bufB;
  bufA = *a;
  bufB = *b;
  *a = bufA / bufB;  //Указатель ссылается на результат
                     //целочисленного деления без остатка
  *b = bufA % bufB;  //Указатель ссылается на остаток от
                     //целочисленного деления
  return bufA + bufB;  //Функция возвращает сумму
}
...
int v1, v2, sum;

v1 = 10;
v2 = 3;
sum = SumAndDiv(&v1, &v2);    //sum = 13; v1 = 3; v2 = 1
```

В функции SumAndDiv вначале сохраняются в буферных переменных значения, на которые указывают указатели a и b. Затем в переменную, на которую указывает указатель a записывается результат деления переданных В функцию значений без остатка, а в переменную, на которую указывается указатель b – остаток от такого деления. Поскольку с помощью указателей мы напрямую обращались к ячейкам памяти, а не к переменным, то после выхода из функции содержимое этих ячеек остается неизменным. При вызове функции SumAndDiV в нее передаются адреса переменных v1 и v2, которым в теле функции соответствуют указател И a и b

**Указатели на структуры**

На структуры можно создавать указатели точно так же, как и для любого другого типа данных. Пример применения такого указателя:

```
struct DATE {
   int Day;
   int Month;
   int Year;
}
...
struct DATE MyBirthday, *dateP;
dateP = &MyBirthday;  //dateP указывает на структуру
MyBirthday
```

В таком случае для доступа к полям структуры через указатель используется оператор

```
dateP->Day = 7;
dateP->Month = 8;
dateP->Year = 1974;
```

Можно также использовать и разыменование указателя:

```
(*dateP).Day = 7;
(*dateP).Month = 8;
(*dateP).Year = 1974;
```

Указатели также могут быть полями структуры. Это часто используется для Создания в памяти цепочек однотипных структур, когда каждая предыдущая указывает на следующую:

```
struct DATE {
  int Dav;
   int Month;
   int Year;
   struct DATE *next_date; //указатель на другую структуру
                           //того же типа
}
...
struct DATE date1, date2;
date1.next_date = &date2;
```

Теперь к примеру, в оператору datel . next_date->Year соответствует доступ у полю Year структуры date2 (то есть, это эквивалентно записи date2.Year)

**Массивы и строки**

Массив - это тип данных, который используется для представления последовательности Однотипных значений. Массивы объявляются подобно обычным переменным, с указанием в квадратных скобках размерности (количества элементов в массиве):

```
int digits[10]; //Массив из десяти элементов типа int
char str[10];   //Массив из десяти символов (строка)
```

Доступ к элементам массива реализуется с помощью индекса (порядкового номера элемента, начиная с О):

```
digits[0] = 0;
digits[1] = 1;
str[0] = 'A';
str[1] = 'B';
```

Зачастую гораздо удобнее инициализировать массив непосредственно при его объявлении, например:

```
int digits[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
char str[10] = {'T', 'h', 'e', ' ', 'l', 'i', 'n', 'e'};
int n;
char c;
n = digits[2];  //n = 2
c = str[1];     //c = 'h'
```

**Строки**

Строка в С - это массив типа char. Выше был рассмотрен пример объявления такого массива, соответствующего строке “The line”. Это объявление можно было бы выполнить и с помощью строкового литерала:

```
char str[10] = "The line";
```

Однако в таком случае следует помнить, что компилятор неявно завершает строковые литералы символом '\0', и, таким образом, реальная длина строки больше на 1. Это следует учитывать, чтобы при инициализации массива не выйти за его пределы.

При инициализации строк размерность массива можно явно не указывать:

```
char str[] = "The line";
```

В этом случае компилятор определит размерность самостоятельно. (Это правило применимо и к инициализации любых других массивов.)

**Многомерные массивы**

Язык С допускает использование многомерных массивов, то есть массивов, элементами которых являются массивы. Примеры объявления таких массивов:

```
int a2[10][2];   //Двухмерный массив 10x2
int a3[3][2][5]; //Трехмерный массив 3x2x5
```

Фактически, все элементы многомерного массива хранятся в памяти последовательно, поэтому представленные ниже примеры инициализации аналогичны:

```
int a[2][3] = { {1, 2, 3},
                {4, 5, 6} };
```

и

```
int a[2][3] = { 1, 2, 3, 4, 5, 6 };
```

Для Доступа к элементам многомерного массива используются индексы по каждой из размерностей или операции разыменования указателя. Например, чтобы извлечь в некоторую переменную n цифру 4 из представленного выше массива a, можно воспользоваться одним из двух вариантов:

```
n = a[1][0];  //Извлекаем элемент из "строки" 1 (вторая),
              //"столбца" 0 (первый), то есть - цифру 4
n = *(a + 3); //a - указатель на первый элемент массива,
              //следовательно a + 3 - указатель на 4-й
элемент
```

**Операторы ветвления**

Операторы ветвления используются для выполнения того или иного блока кода в зависимости от некоторого условия. К таким операторам в языке С относятся `if-else` и `switch-case`.

**Оператор if-else**

В простейшем случае оператор if -else имеет следующую структуру:

```
if (условное_выражение) блок_кода_1; else блок_кода_2;
```

Если условное выражение истинно, то выполняется блок кода 1, в противном случае - блок кода 2. При этом в качестве блока кеда 2 допускается использовать последовательность операторов else-if

```
if (выражение1) блок_кода_1;
else if (выражение2) блок_кода_2;
     else if (выражение3) блок_кода_3; ... else блок_кода_N;
```

В данном случае каждое выражение будет вычисляться поочередно до тех пор, пока не будет найдено выражение, давшее истинный результат. Если же результаты вычислений всех выражений окажутся ложными, то будет выполнен блок кода N.

В тех случаях, когда при ложных результатах всех условных выражений не требуется выполнять никаких операторов, можно опустить завершающий блок кода вместе с последней ветвью else. Например:

```
if(a == 1) b = a*3;
else if(a == 2) b = a + 10;
     else if(a == 3) b = 0;
```

Если выражение в первой строке будет истинным, остальные операторы будут пропущены, в противном случае начнется последовательная проверка выражений, указанных в следующих строках, до тех пор, пока не будет найдено выражение, дающее ненулевой результат, или до тех пор, пока не будет проверено последнее выражение. Если ни одно из выражений не будет истинным, никакие действия с переменной b выполнены не будут.

**Условные выражения**

Язык С позволяет вместо оператора if-else использовать условные выражения. Так, конструкцию вида

```
if (условие) блок_кода_1; else блок_кода_2;
```

можно заменить следующим условным выражением:

```
условие ? блок_кода_1 : блок_кода 2;
```

Например:

```
(a == 1) ? b = a*3 : b = 0; //Если a=1, то b=a*3, иначе b=0
```

**Оператор switch**-case

В оператора if-else можно использовать только выражения, которые сводятся к значению TRUE: или FALSE. В тех случаях, когда необходимо применять выражения, дающие произвольный числовой результат, удобнее воспользоваться оператором switch-case. Этот оператор позволяет с помощью некоторой переменной выбирать ОДНО из выражений, соответствующее заданной константе. Его синтаксис:

```
switch (выражение) {
   case константа-выражение1 : блок_кода
   case константа-выражение2 : блок_кода
   case константа-выражение3 : блок_кода
   default: блок_кода
```

Продемонстрируем использование оператора switch-case еа примере рассмотренного выше фрагмента, реализованного с помощью оператора if-else:

```
switch (a) {
   case 1 : b = a * 3; break;
   case 2 : b = a + 10; break;
   case 3 : b = 0; break;
default:
}
```

Оператор break приводит к немедленному вых0ду из блока switch. Если по каким-то причинам необходимо продолжить проверку соответствия выражения выбора тем константам, которые указаны в ветвях case, следует убрать операторы break.

Ко всему прочему, ветви case можно каскадировать. Такой прием особенно удобен в тех случаях, когда нужно, например, проверить введенный символ, не заботясь о том, представляет ли он прописную букву или строчную, или когда нужно получить одну и ту же реакцию на несколько разных чисел. Рассмотрим это на примере фрагмента некоторой абстрактной

```
switch (input) {
   case 'a' : case 'A' : DoA(); break;
   case 'b' : case 'B' : DoB(); break;
   case '0' : case '1' : case '2' : case '3' : Do0123();
break;
   case '4' : case '5' : case '6' : case '7' : D04567();
break;
   default : DoDefault(); break;
}
```

**Циклические конструкции**

Циклические конструкции применяют для повторения некоторого блока кода на основании условия цикла. В языке С используются циклические конструкции `while`, `for` и `do-while`.

**Конструкция while**

Цикл while имеет следующий синтаксис.

```
while (условное_выражение)
{
   // Выполнение тела цикла, если выражение истинно
}
```

Другими словами, циклы while имеет смысл использовать в тех случаях, когда соответствующий оператор или блок операторов необходимо выполнять до тех пор, пока условное выражение истинно. Пример формирования строки, состоящей из нечетных цифр:

```
int c, i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
c = 0; //Счетчик циклов
i = 0; //Индекс массива OddNums
while (c < 10) //До тех пор, пока с меньше 10,...
{
  //Если остаток от деления с на 2 = 1, то записываем в i-ю
  //позицию массива OddNums с-й элемент строки str, после
чего
  //значение i автоматически инкрементируется
  if ((c % 2) == 1) OddNums[i++] = str[c];
  c++; //c = c + 1
}
```

**Конструкция for**

Цикл for имеет следующий синтаксис.

```
for (выражение1; выражение2; выражение3)
{
   // Выполнение тела цикла
}
```

Выражение1 выполняется только один раз при входе в цикл, и обычно представляет собой оператор присваивания некоторого начального значения счетчику цикла. Выражение 2 - это условное выражение, определяющее момент выхода из цикла (цикл выполняется до тех пор, пока оно равно TRUE или 1). Выражение 3 еще один оператор присваивания, в котором обычно изменяется счетчик цикла или некоторая переменная, влияющая на выполнение условия в выражении2. Выражения могут быть представлены любыми операторами, включая пустые (то есть, вместо выражения можно поставить только символ точки с запятой).

Циклы while и for В большинстве случаев взаимозаменяемы. Так, представленный выше пример ДЛЯ цикла while, можно переписать в следующем вице:

```
int c, i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
i = 0; //Индекс массива OddNums
for (c = 0; c < 10; c++) //До тех пор, пока с меньше 10,...
  if ((c % 2) == 1) OddNums[i++] = str[c];
```

В одних ситуациях удобнее применять циклы f or, В других - while. Достоинством циклов for является более наглядная инициализация и организация изменения счетчика цикла. С другой стороны, циклы while более гибкие и обеспечивают больше возможностей для реализации нестандартных программных решений при организации повторяющихся вычислений.

**Конструкция do~while**

Кроме циклов while и for, В которых вначале выполняется проверка истинности условия цикла, и только потом управление передается блоку операторов цикла, в языке С имеется также конструкция do-while. Она отличается от первых двух тем, что в ней вначале выполняется блок операторов, и только потом проверяется выполнение условия. Другими словами, цикл do-while всегда выполняется как минимум один раз, вне зависимости от условия цикла.

Цикл do-while имеет следующий синтаксис.

```
do
{
   // Выполнение блока операторов цикла
}
while (условное_выражение);
```

**Организация бесконечных циклов**

Для организации бесконечного цикла в качестве условного выражения в конструкции while или do-while можно просто указать значение TRUE или 1:

```
while(1) блок_операторов;
```

В случае циклов for это будет выглядеть следующим образом:

```
for (;;) блок_операторов;
```

**Операторы break u continue**

Если в теле любого цикла встречается оператор break, управление тут же передается на оператор, следующий за оператором цикла, вне зависимости от истинности или неистинности условного выражения. При этом во вложенных циклах выход осуществляется не на самый верхний уровень вложенности, а лишь на один уровень вверх,

При выполнении оператора continue все находящиеся после него операторы блока пропускаются, а управление передается в начало цикла для следующей итерации. На практике операторы continue используются гораздо реже, чем операторы break, однако в сложных циклах, требующих принятия решений на основании многих факторов, использование операторов continue может быть весьма удобным.

**Стандартные функции ввода/вывода**

Стандартные функции ввода/вывода позволяют обмениваться информацией с выполняемой программой, что, к примеру, очень удобно в процессе отладки. Все они построены на основе функций посимвольного ввода/вывода, которые легко приспособить к аппаратным требованиям системы. Если при обычном применении языка С эти функции используются для ввода данных с клавиатуры и вывода на экран или принтер, то применительно к микроконтроллерам AVR и PIC речь идет об обмене данными через приемопередатчик UART/USART (no умолчанию) или последовательный порт. Таким образом, прежде., чем начать ВВОД/ВЫВОД
в программе должны быть выполнены соответствующие настройки (в частности, в компиляторе CCS-PICC для этой цели используется директива препроцессора use, о чем речь пойдет чуть позже).

**Ввод/вывод символов с помощью функций getchar () и putchar()**

Простейшими функциями ввода/вывода в языке С являются getchar() и putchar(), которые предназначены для посимвольного обмена данными (обе объявлены в стандартном заголовочном файле stdio.h). Функция getchar() возвращает символ, принятый от UART/USART или по последовательному интерфейсу, а функция putchar ( ) , наоборот, выводит символ. Все остальные стандартные функции ввода/вывода базируются на них.

Рассмотрим пример простой программы, выводящей в бесконечном цикле через приемопередатчик UART/USART символы “1”, “2”, “3”, “4” и “5”. Исходный код программы putchar. с для микроконтроллеров AVR представлен в листинге. Пример - для устройства AT90585 1 5.

**Листинг 3.1. Программа `putchar.c` для микроконтроллеров AVR**

```
#include <avr/io.h>
#include <stdio.h>

int main (void)
{
  const char str[] = "12345";
  int i;
  UBRR = 25;    //Скорость обмена через UART - 9600 бод
                //(см. табл. 1.17)
  UCR = 0x18;   //Устанавливаем разряды TXEN и RXEN для
                //активизации UART в режиме ввода/вывода через
                //выводы 0 и 1 порта D.
                //Разряд CHR9=0 - передача 8 бит данных
  i = 0;        //Индекс строки символов
  while (1)     //Бесконечный цикл
  {
    if (i > 4) i = 0; //Проверка выхода за пределы массива
    putchar(str[i]);  //Выводим i-й символ через UART
  }
}
```

Для микроконтроллеров PIC этот же пример будет выглядеть несколько иначе (см. листинг). В компиляторе CCS-PICC для настройки частоты системной синхронизации и параметров обмена данными через

Листинг 3.2. Программа putchar.c для микроконтроллеров PIC

```
#include <18F458.h>
#fuses HS, NOWDT
#use delay(clock=10000000)
#use
rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,stream=RS232,bits=8
)

void main()
{
  const char str[] = "12345";
  int i;
  i = 0;
  while (1)
  {
    if (i > 4) i = 0;
    putchar(str[i]);
  }
}
```

## Функции вывода строк puts () и printf()

Функции puts ( ) и printf () используют функцию put char ( ) для вывода посимвольно строки в порт R8232, назначенным последним. Предположим, у насесть определение строки, предназначенной для вывода через последовательный интерфейс:

```
char s[6] = "12345";
```

Вызов функции puts ( ) для вывода этой строки выглядит как puts(s);.

Функция pr intf ( ) несколько сложнее, поскольку позволяет применить форматированный вывод. Она имеет следующий синтаксис вызова:

```
printf("Строка, в которую подставляются переменные ",
        переменная1, переменная2, …. );
```

В выводимой строке обычно используются спецификации форматирования значений переменных, переданных в качестве параметров. Каждая такая спецификация начинается со знака “%”, после которого следуют обозначения параметров форматирования:

- **c** — вывод параметра в виде символа ASCII;
- **d** — вывод параметра в виде целого числа;
- **e** — вывод параметра в экспоненциальном формате;
- **f** — вывод параметра в виде вещественного числа;
- **ld** — вывод параметра в виде длинного целого со знаком;
- **lu** — вывод параметра в виде беззнакового длинного целого;
- **Lx** —вывод параметра в виде беззнакового длинного целого в шестнадцатеричном представлении, используя нижний регистр символов;
- **LX** —вывод параметра в виде беззнакового длинного целого в шестнадцатеричном представлении, используя верхний регистр символов;
- **s** — вывод параметра в виде строки;
- **u** — вывод параметра в виде беззнакового целого;
- **x** —вывод параметра в виде беззнакового целого в шестнадцатеричном представлении, используя нижний регистр символов;
- **X** —вывод параметра в виде беззнакового целого в шестнадцатеричном представлении, используя верхний регистр символов;
- **%** — вывод знака процента.

Количество спецификаций должно совпадать с количеством переменных. При этом первая встретившаяся спецификация соответствует первой переменной в списке, вторая - второй и т.д.

В случае вывода числовых значений сразу же после знака “%” может быть указано количество символов и формат для отображения чисел, например:

8 — восемь знаков;

08 — восемь знаков с дополнением ведущими нулями;

8.2 — восемь знаков, два знака после десятичной точки.

Примеры использования спецификаций форматирования:

```
int n = 123;
 char c = '$';
 float f = 23.5;
 char str[] = "Hello";
 printf("n = %d, str = %s", n, str);  //n = 123, str = Hello
 printf("n = %d : 0x%04X", n, n);     //n = 123:0x007B
 printf("Salary = %c%8.2f", c, f);    //Salary = $23.50
```

**Функции ввода строк gets () и scanf()**

Функции gets ( ) и scanf ( ) выполняют действия, обратный функциям puts () и printf ( ) : считывают посимвольно строку через назначенный последним последовательный интерфейс.

Рассмотрим пример использования функций gets () и puts ( ) для микроконтроллеров PIC ( см.листинг).

**Листинг 3.3. Программа gets.c для микроконтроллеров PIC**

```
#include <18F458.h>
#fuses HS, NOWDT
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

char s1[12]; //Буфер, в который будет помещена принятая строка

void main()
{
  while (1)
  {
    gets(s1);  //Считываем строку из стандартного входного потока
    puts(s1);  //Выводим строку в стандартный выходной поток
  }
}
```

Функция scanf ( ) считывает из вх0дного потока значения в формате, определенном в первом параметре, и сохраняет их в переменных, адреса которых переданы ей в качестве последующих параметров. Спецификации форматирования для этой функции такие же, как и для printf ( ) . Обычно функция scanf ( ) используется в паре с printf

```
int x, y;
puts("Enter two integers: ");
scanf("%d,%x\n", &x, &y); //Считывает два числа, введенные
                          //через запятую: одно - в
десятичной,
                          //а другое - в шестнадцатеричной
форме
printf("x = %d, y = %04X", x, y);
```

Как уже было отмечено ранее, директивы препроцессора, по сути, не являются составной частью языка С, а используются для подстановки коца в текст программы. Препроцессор - это особая программа, которая выполняет предварительную обработку данных до начала компиляции. Рассмотрим стандартные директивы препроцессора, а также директивы, характерные для компилятора CCS-PICC.

**Директива #include**

Выше, в разделе “Структуры программы на С” уже упоминалась самая распространенная директива #include, которая используется фактически во всех программах для включения в текст программы заголовочных файлов (с определениями), имеющий расширение .1'1, или файлов .c (не содержащих функцию main())

Эта директива может использоваться в двух формах:

```
#include <имя_файла>
```

или

```
#include "имя_файла"
```

Если имя файла заключено между знаками “<” и “>”, то он считается частью стандартной библиотеки, поставляемой вместе с компилятором. Если же имя файла заключено в двойные кавычки, то считается, что он расположен в той же папке, что и файл с исх0дным кодом.

**Директива #define**

Директива #define указывает препроцессору на необх0димость выполнить п0дстановку в тексте программы определенной последовательности символовдругой последовательностью. Формат директивы:

```
#define заменяемая_последовательность фрагмент_подстановки
```

Например:

```
#define MyName "John Smith"
#define Condition (a > b)
#define Operation a = b
…
char str[] = MyName; //Равнозначно char str[] = "John Smith";
int a, b;
if Condition Operatrion; //Равнозначно if (a > b) a = b;
```

В директиве #define могут использоваться параметры, благодаря чему она становится очень мощный и гибким инструментом, позволяющим заменять один простой текстовый элемент сложным выражением. Такие подстановки называют макросами.

Например, выражение, В котором выбирается большее из двух значений, можно представить в виде следующего макроса.

```
#define larger(x, y) ( (x)>(y) ? (x) : (y) )
```

Если определен такой макрос, то код, использующий его, может иметь следующий вид.

```
int a = 9;
int b = 7;
int c = 0;
c = larger(a, b);
```

Напоминает вызов функции, Однако это не функция - компилятор получит от препроцессора последнюю строку в следующем вице.

```
c = ( (a)>(b) ? (a) : (b) );
```

Основное отличие макроса от функции заключается в том, что код макроподстановки вставляется препроцессором в программный код везде, где встречается заданный текстовый элемент, код же функции определяется только в одном месте, а в тех местах, где указано ее имя, осуществляется вызов этого кода.

Есть и еще одно отличие, особенно важное при программировании микроконтроллеров, - при использовании макросов, в отличие от функций, ничего не помещается в стек, что позволяет сэкономить оперативную память.

Кроме того, макросы преобразуются в обычный код, который выполняется быстрее, чем код функции, на вызов которого и возврат управления в вызывающую функцию ух0дит дополнительное машинное время. Наконец, при использовании макросов не требуется формального объявления типов данных.

Перечислим некоторые правила использования директивы #de fine:

- при создании комментариев в строке с #define всегда используется комментарий вида /* ... */;
- следует помнить, что конец строки - это конец #define, и весь текст слева заменить текст справа;
- для преобразования параметра макроса в текстовую строку можно указать перед ним символ “#”, например:

```
#define OutString(s) puts(#s)
OutString(Line); //Равнозначно puts("Line");
```

- для конкатенации двух параметров можно воспользоваться оператором например:

```
#define Concat(x, y) x ## y
int i = Concat(2,1); //Равнозначно int i = 21;
```

- для переноса текста подстановки на другую строку используется символ обратной косой “\”, например:

```
#define LongStr "0 1 2 3 4 5 6 7 8 9 10 \
11 12 13 14 15 16 17 18 19 20 "
```

- для отмены определения используется директива #unde f, например:

```
#define A_Char 'A'
…
#undef A_Char
#define A_Char 'a'.
```

Многие микроконтроллеры отличаются лишь некоторыми параметрами, количеством выводов, размером памяти и размещением регистров. Это позволяет создавать на языке С программный код для всего модельного ряла.` Однако для этого следует каким-то образом заменить те параметры, которые отличаются у разных моцелей. Для таких целей используются директивы условной компиляции `#ifdef`, `#ifndef`, `#else` и `#endif`, а также `#if` и `#elif`.

Синтаксис для директивы #ifdef

```
#ifdef имя_макроса
   последовательность_операторов_1
#else
   последовательность_операторов_2
#endif
```

Если имя макроса определено в программе, то компилируется первая последовательность операторов, в противном случае - вторая последовательность (ветка #else может и отсутствовать).

Пример использования (для компилятора CCS-PICC):

```
#define DEBUGGING_ON
#ifdef DEBUGGING_ON
   #use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#endif
```

Синтаксис для директивы #ifnde f

```
#ifndef имя_макроса
   последовательность_операторов_1
#else
   последовательность_операторов_2
#endif
```

В данном случае, в отличие от директивы #ifde f, первая последовательность операторов выполняется в том случае, если имя макроса в программе не определено.

Для условной компиляции можно также воспользоваться директивами #if, elif. Их синтаксис:

```
#if выражение1
   последовательность_операторов_1
#elif выражение2
   последовательность_операторов_2
#else
   последовательность_операторов_3
#endif
```

Эта конструкция работает аналогично условному оператору if “else. Компилятор оценивает выражения после #if и #elif до тех пор, пока Одно из них не даст в результате TRUE, после чего в текст программы подставляется соответствующая последовательность операторов. Если оба выражения дают

FALSE, то подставляется последовательность операторов после директивы #else (если она присутствует).

Допустим, для передачи и приема данных через UART у абстрактного микроконтроллера SomeMic16 используются выводы 12 и 13, у микроконтроллера SomeMic8 - выводы 6 и 14, а у SomeMic4 - выводы 1 и 2. Тогда мы можем создать заголовочный файл SomeMic . h и включить в него следующие директивы.

```
#if SomeMicX == 16
   #define TXD 12
   #define RXD 13
#elif SomeMicX == 8
   #define TXD 6
   #define RXD 14
#elif SomeMicX == 4
   #define TXD 1
   #define RXD 2
#else
   #error "Pins TXD и RXD for SomeMicX are not defined"
#endif
```

Теперь, если программный проект будет построен на микроконтроллере SomeMic8, то в заголовочный файл следует поместить следующий текст.

```
#ifndef SomeMicX
   #define SomeMicX = 8
   #include <SomeMic.h>
#endif
```

Встретив директиву #ifnde f, препроцессор включит В текст программы define SomeMicX == 8 и #include <SomeMic.h>. Поскольку после этого элемент SomeMicX получит значение, равное 8, то любая повторная обработка директивы #ifndef не приведет к дублированию в выходном тексте соответствующей информации. Другими словами, c0держимое заголовочного файла SomeMic.h будет помещено в исходный код файлов, использующих заголовочный файл с ifndef, только один раз.

Использование директивы #ifndef с последующими директивами #de fine и include - стандартный прием, позволяющий избежать дублирования информации из заголовочных файлов в исходном коде проекта. Если этого не сделать, то при дублировании компилятор обычно выдает множество сообщений об ошибках, связанных с многократным объявлением переменных.

**Директива #error**

Директива #еггог используется сон:.iестно с директивами условной компиляции. Встретив ее, компилятор сгенерирует сообщение об ошибке, указанное справа от директивы (см. пример в предыдущем подразделе).

**Директивы, характерные для компилятора CCS-PICC**

Компилятор CCS-PICC поддерживает множество встроенных, нестандартных директив. Рассмотрим некоторые из них.

**Директива #bit**

Директива препроцессора #bit используется для получения доступа к отдельным разрядам регистров или переменных. Ее синтаксис:

```
#bit идентификатор = x.y
```

где x - константа, определяющая регистр, или переменная; y - константа от 0 до 7. Например, следующим образом переменная TOIF объявляется как разряд 2 регистра по адресу 0x08:

```
#bit T0IF = 0x0B.2
```

Пример для разряда переменной:

```
inc c;
#bit CBit0 = c.0;
…
if (CBit0 ==0) …
```

**Директива #byte**

Директива #byte имеет следующий синтаксис:

```
#byte идентификатор = X
```

где X - имя переменной или константы.

Если идентификатор - это уже объявленное ранее имя переменной, то компилятор помещает эту переменную по адресу X, В противном случае компилятор создает новую переменную и помещает ее по адресу X как 8-разрядное целое число. В обоих случаях в ячейку памяти с адресом X могут быть помещены любые другие переменные. Фактически, если X - имя переменной, то ей будет соответствовать тот же адрес в памяти, что и для объявленного идентификатора:

```
char c; //Переменная c размещается в памяти компилятором
#byte a = c; //Переменной b назначен тот же адрес, что и c
```

**Директива #case**

Директива препроцессора #case указывает компилятору быть чувствительным к регистру символов. По умолчанию, компилятор CCS-PICC не чувствителен к регистру символов.

# 4.SCADA-**СИСТЕМЫ**

Современная АСУТП (автоматизированная система управления технологическим процессом) представляет собой многоуровневую человеко-машинную систему управления. Создание АСУ сложными технологическими процессами осуществляется с использованием автоматических информационных систем сбора данных и вычислительных комплексов, которые постоянно совершенствуются, по мере эволюции технических средств и программного обеспечения.

*АСУТП и диспетчерское управление*

Непрерывную во времени картину развития АСУТП можно разделить на три этапа, обусловленные появлением качественно новых научных идей и технических средств. В ходе истории меняется характер объектов и методов управления, средств автоматизации и других компонентов, составляющих содержание современной системы управления.

• Первый этап отражает внедрение систем автоматического регулирования (САР). Объектами управления на этом этапе являются отдельные параметры, установки, агрегаты. Решение задач стабилизации, программного управления, слежения переходит от человека к САР. У человека появляются функции расчета задания и параметров настройки регуляторов.

• Второй этап - автоматизация технологических процессов. Объектом управления становится рассредоточенная в пространстве система. С помощью систем автоматического управления (САУ) реализуются все более сложные законы управления, решаются задачи оптимального и адаптивного управления, проводится идентификация объекта и состояния системы. Характерной особенностью этого этапа является внедрение систем телемеханики в управление технологическими процессами. Человек все больше отдаляется от объекта управления, между объектом и диспетчером выстраивается целый ряд измерительных систем, исполнительных механизмов, средств телемеханики, мнемосхем и других средств отображения информации (СОИ).

• Третий этап - автоматизация систем управления технологическими процессами - характеризуется внедрением в управление технологическими процессами вычислительной техники. Вначале - применение микропроцессоров, использование на отдельных фазах управления вычислительных систем; затем - активное развитие человеко-машинных систем управления, инженерной психологии, методов и моделей исследования операций и, наконец, - диспетчерское управление на основе автоматических информационных систем сбора данных и современных вычислительных комплексов.

От этапа к этапу меняются и функции человека (оператора/диспетчера), призванного обеспечить регламентное функционирование технологического процесса. Расширяется круг задач, решаемых на уровне управления. Ограниченный прямой необходимостью управления технологическим процессом набор задач пополняется качественно новыми задачами, ранее

имеющими вспомогательный характер или относящимися к другому уровню управления. Диспетчер в многоуровневой автоматизированной системе управления технологическими процессами получает информацию с монитора ЭВМ или с электронной системы отображения информации и воздействует на объекты, находящиеся от него на значительном расстоянии, с помощью телекоммуникационных систем, контроллеров, интеллектуальных исполнительных механизмов. Основой, необходимым условием эффективной реализации диспетчерского управления, имеющего ярко выраженный динамический характер, становится работа с информацией, т.е. процесс сбора, передачи, обработки, отображения, представления информации.

От диспетчера уже требуется не только профессиональное знание технологического процесса, основ управления, но и опыт работы в информационных системах, умение принимать решение (в диалоге с ЭВМ) в нештатных и аварийных ситуациях и многое другое. Диспетчер становится главным действующим лицом в управлении технологическим процессом.

Говоря о диспетчерском управлении, нельзя не затронуть проблему технологического риска. Технологические процессы в энергетике, нефтегазовой и ряде других отраслей промышленности являются потенциально опасными и при возникновении аварий приводят к человеческим жертвам, а также к значительному материальному и экологическому ущербу.

Статистика говорит, что за тридцать лет (с начала 60-х до конца 80-х годов XX века) число учтенных аварий удваивалось примерно каждые десять лет. В результате анализа большинства аварий и происшествий на всех видах транспорта, в промышленности и энергетике были получены интересные данные. В 60-х годах ошибка человека была первоначальной причиной аварий лишь в 20% случаев, тогда как к концу 80-х доля «человеческого фактора» стала приближаться к 80%.

Одна из причин этой тенденции - старый, традиционный подход к построению сложных систем управления, т. е. ориентация на применение новейших технических и технологических достижений и недооценка необходимости построения эффективного человеко- машинного интерфейса, ориентированного на человека (диспетчера). Таким образом, требование повышения надежности систем диспетчерского управления является одной из предпосылок появления нового подхода при разработке таких систем. Основа современного подхода - ориентация на оператора/диспетчера и его задачи. Концепция SCADA (Supervisory Control And Data Acquisition - диспетчерское управление и сбор данных) предопределена всем ходом развития систем управления и результатами научно-технического прогресса. Применение SCADA-технологий позволяет достичь высокого уровня автоматизации в решении задач разработки систем управления, сбора, обработки, передачи, хранения и отображения информации.

Дружественность человеко-машинного интерфейса (HMI/MMI - Humain/Man Machine Interface), предоставляемого SCADA-системами, полнота и наглядность представляемой на экране информации, доступность «рычагов»

управления, удобство пользования подсказками и справочной системой и т. д. повышают эффективность взаимодействия диспетчера с системой и сводят к минимуму его критические ошибки при управлении. Следует отметить, что концепция SCADA, основу которой составляет автоматизированная разработка и управление в реальном времени, позволяет решить еще ряд задач, долгое время считавшихся неразрешимыми: сокращение сроков разработки проектов по автоматизации и прямых финансовых затрат на их разработку.

В настоящее время SCADA является основным и наиболее перспективным методом автоматизированного управления сложными динамическими системами (процессами).

Управление технологическими процессами на основе систем SCADA стало осуществляться в передовых западных странах в 80-е годы. Область их применения охватывает сложные объекты электро- и водоснабжения, химические, нефтехимические и нефтеперерабатывающие производства, железнодорожный транспорт, транспорт нефти и газа и др.

В России диспетчерское управление технологическими процессами опиралось, главным образом, на опыт оперативно-диспетчерского персонала. Переход к управлению на основе SCADA-систем стал осуществляться позднее. К трудностям освоения в России новой информационной технологии - SCADA-систем - относятся как отсутствие эксплуатационного опыта, так и недостаток информации о различных SCADA-системах. В мире насчитываются не один десяток компаний, активно занимающихся разработкой и внедрением SCADA-систем. Каждая SCADA-система - это ноу-хау компании, и поэтому информация о той или иной системе не столь обширна.

Большое значение при внедрении современных систем диспетчерского управления имеет решение следующих задач:

• выбор SCADA-системы (исходя из требований и особенностей технологического процесса);

• кадровое сопровождение.

Выбор SCADA-системы представляет собой достаточно трудную задачу, аналогичную принятию решений в условиях многокритериальности, усложненную невозможностью количественной оценки ряда критериев из-за недостатка информации. Подготовка специалистов по разработке и эксплуатации систем управления на базе программного обеспечения SCADA осуществляется на специализированных курсах различных фирм, курсах повышения квалификации. В настоящее время в учебные планы ряда технических университетов начали вводиться дисциплины, связанные с изучением SCADA-систем. Однако специальная литература по SCADA-системам отсутствует, имеются лишь отдельные статьи и рекламные проспекты.

*Компоненты систем контроля и управления и их назначение*

Многие проекты автоматизированных систем контроля и управления (СКУ) для большого спектра областей применения позволяют выделить обобщенную схему их реализации, представленную на рис.4. 1.

Операторские станции
SCADA
Ethernet
SQL Server
Расчетные станции
Интеллектуальный контроллер
Офисная сеть Ethernet
MnKpoSCADA
Промышленная сеть
Контроллеры нижнего уровня

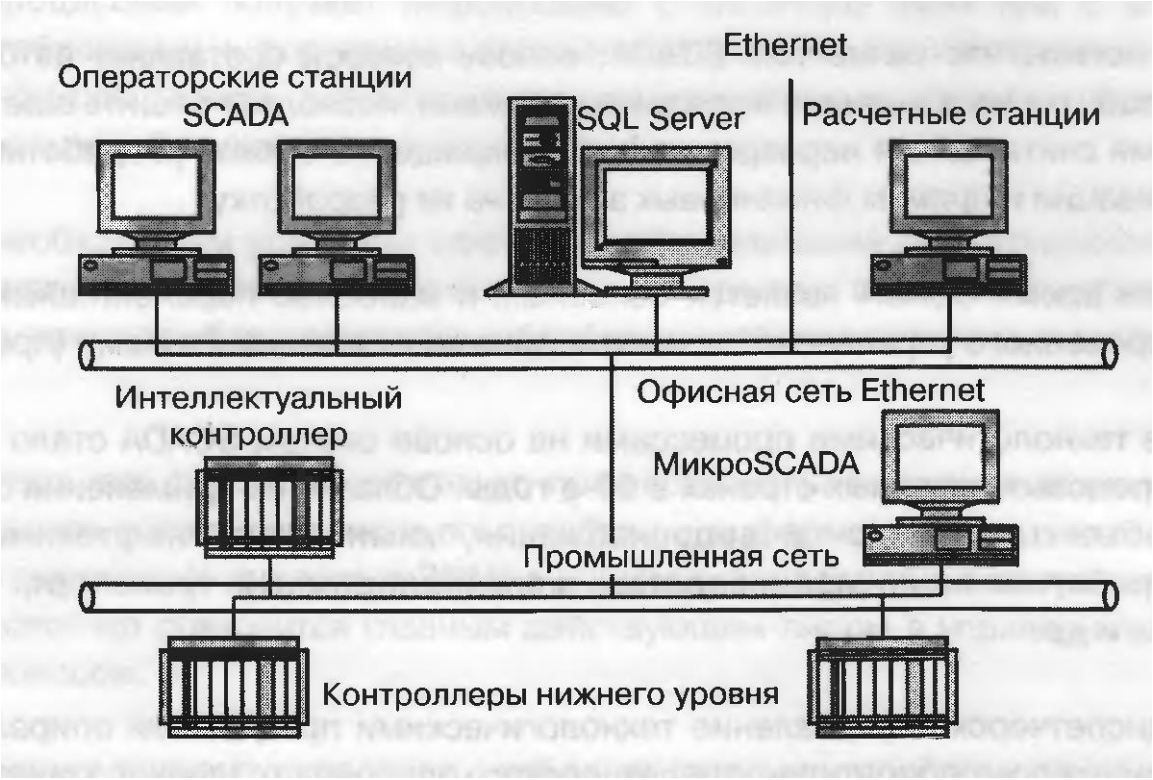



Рис. 4.1. Обобщенная схема системы контроля и управления

Как правило, это двухуровневые системы, так как именно на этих уровнях реализуется непосредственное управление технологическими процессами. Специфика каждой конкретной системы управления определяется используемой на каждом уровне программно-аппаратной платформой.

• Нижний уровень - уровень объекта (контроллерный) - включает различные датчики для сбора информации о ходе технологического процесса, электроприводы и исполнительные механизмы для реализации регулирующих и управляющих воздействий. Датчики поставляют информацию локальным программируемым логическим контроллерам (PLC - Programming Logical Controller), которые могут выполнять следующие функции:

- сбор и обработку информации о параметрах технологического процесса;
- управление электроприводами и другими исполнительными механизмами;

- решение задач автоматического логического управления и др.

Так как информация в контроллерах предварительно обрабатывается и частично используется на месте, существенно снижаются требования к пропускной способности каналов связи.

В качестве локальных PLC в системах контроля и управления различными технологическими процессами в настоящее время применяются контроллеры как отечественных, так и зарубежных производителей. На рынке представлены сотни типов контроллеров, способных обрабатывать от нескольких десятков до нескольких десятков тысяч переменных.

К аппаратно-программным средствам контроллерного уровня управления предъявляются жесткие требования по надежности, времени реакции на исполнительные устройства, датчики и т.д. Программируемые логические контроллеры должны гарантированно откликаться на внешние события, поступающие от объекта, за время, определенное для каждого события.

Для критичных с этой точки зрения объектов рекомендуется использовать контроллеры с операционными системами реального времени (ОСРВ). Контроллеры под управлением ОСРВ функционируют в режиме жесткого реального времени.

Разработка, отладка и исполнение программ управления локальными контроллерами осуществляется с помощью специализированного программного обеспечения (ПО), широко представленного на рынке.

К этому классу инструментального ПО относятся пакеты типа ISaGRAF (CJ International France), InConrol (Wonderware, USA), Paradym 31 (Intellution, USA), имеющие открытую архитектуру.

Информация с локальных контроллеров может направляться в сеть диспетчерского пункта непосредственно, а также через контроллеры верхнего уровня. В зависимости от поставленной задачи контроллеры верхнего уровня (концентраторы, интеллектуальные или коммуникационные контроллеры) реализуют различные функции. Некоторые из них перечислены ниже:

- сбор данных с локальных контроллеров;
- обработка данных, включая масштабирование;
- поддержание единого времени в системе;
- синхронизация работы подсистем;
- организация архивов по выбранным параметрам;
- обмен информацией между локальными контроллерами и верхним уровнем;
- работа в автономном режиме при нарушениях связи с верхним уровнем;
- резервирование каналов передачи данных и др.

Верхний уровень - диспетчерский пункт (ДП) - включает, прежде всего, одну или несколько станций управления, представляющих собой автоматизированное рабочее место (АРМ) диспетчера/оператора. Здесь же могут быть размещены: сервер базы данных, рабочие места (компьютеры) для специалистов и т. д. Часто в качестве рабочих станций используются ПЭВМ типа IBM PC различных конфигураций.

Станции управления предназначены для отображения хода технологического процесса и оперативного управления. Эти задачи и призваны решать SCADA-системы. SCADA - это специализированное ПО, ориентированное на обеспечение интерфейса между диспетчером и системой управления, а также коммуникацию с внешним миром.

Спектр функциональных возможностей определен самой ролью SCADA в системах управления и реализован практически во всех пакетах: автоматизированная разработка, дающая возможность создания ПО системы автоматизации без реального программирования;
средства исполнения прикладных программ;
сбор первичной информации от устройств нижнего уровня;
обработка первичной информации;
регистрация алармов и исторических данных;
хранение информации с возможностью ее постобработки (как правило, реализуется через интерфейсы к наиболее популярным базам данных);
визуализация информации в виде мнемосхем, графиков и т.п.;
возможность работы прикладной системы с наборами параметров, рассматриваемых как «единое целое» (recipe или «установки»).

Рассматривая обобщенную структуру систем управления, следует ввести и еще одно понятие - Micro-SCADA. Micro-SCADA - это системы, реализующие стандартные (базовые) функции, присущие SCADA-системам верхнего уровня, но ориентированные на решение задач автоматизации в определенной отрасли (узкоспециализированные). В противоположность им SCADA-системы верхнего уровня являются универсальными.

Все компоненты системы управления объединены между собой каналами связи.

Обеспечение взаимодействия SCADA-систем с локальными контроллерами, контроллерами верхнего уровня, офисными и промышленными сетями возложено на так называемое коммуникационное ПО. Это достаточно широкий класс программного обеспечения, выбор которого для конкретной системы управления определяется многими факторами, в том числе и типом применяемых контроллеров, и используемой SCADA-системой.

Большой объем информации, непрерывно поступающий с устройств ввода/вывода систем управления, предопределяет наличие в таких системах баз данных (БД).

Основная задача баз данных - своевременно обеспечить пользователя всех уровней управления требуемой информацией. Но если на верхних уровнях АСУ эта задача решена с помощью традиционных БД, то этого не скажешь об уровне АСУТП. До недавнего времени регистрация информации в реальном времени решалась на базе ПО интеллектуальных контроллеров и SCADA-систем. В последнее время появились новые возможности по обеспечению высокоскоростного хранения информации в БД.

# II.ПРАКТИЧЕСКАЯ ЧАСТЬ

# 1.ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРОГРАММИРОВАНИЮ ПЛК

## 1.1 Общие положения

### ТРЕБОВАНИЯ К СОДЕРЖАНИЮ ОТЧЕТА

1. Титульный лист
2. Название лабораторной работы
3. Цель
4. Задание
5. Структурная схема системы управления
6. Схема подключения
7. Схема алгоритма управляющей программы
8. Листинг программы

### МЕРЫ БЕЗОПАСНОСТИ

При эксплуатации контроллера необходимо строго соблюдать требования пожарной безопасности в соответствии с ГОСТ 12.1.004-76, требований электробезопасности в соответствии с ГОСТ 12.1.019-80, а также общие требования безопасности в соответствии с ГОСТ 12.2.003-74.

Перед подключением ПЛК к сети напряжением 220 В, корпуса блоков процессоров, ввода - вывода и пульта программирования и диагностики должны быть соединены с контуром заземления медной шиной или проводом. Сопротивление заземления между болтом заземления и корпусом ПК должно быть не более 0,1 Ом.

К работе с ПЛК допускаются лица, прошедшие инструктаж по технике безопасности на рабочем месте в лаборатории.

Не разрешается включать ПЛК без разрешения преподавателя.

Запрещается эксплуатировать ПЛК при отсутствии или неисправности заземления, при открытых крышках и снятых кожухах.

Не разрешается касаться одновременно корпуса ПЛК и корпусов других электроприборов.

Не следует во время работы ПЛК отключать кабели, соединяющие между собой отдельные составные части.

Студентам запрещается выполнять какие-либо ремонтные работы ПЛК. Выполнение лабораторной работы рекомендуется проводить бригадами в составе 2 - 3 студентов.

## 1.2 ЛАБОРАТОРНЫЕ РАБОТЫ №1-№17

## ЛАБОРАТОРНАЯ РАБОТА №1

### Ознакомление с основами программирования ПЛК в пакете FX Trainer

**Цель работы**

1. Ознакомиться с обучающим пакетом программирования контроллеров на языке релейно-контактных схем (LD) FX Trainer.
2. Рассмотреть предложенные примеры использования ПЛК в различных отраслях народного хозяйства.

**Краткий обзор обучающего пакета FX Trainer**

Пакет FX Trainer разработан фирмой Mitsubishi Electric с целью быстрого и эффективного обучения программированию ПЛК на языке LD. В процессе инсталляции программного обеспечения устанавливаются все необходимые для изучения составляющие:

- инструменты программирования;
- виртуальный PLC;
- имитатор оборудования;
- экранные переключатели входов-выходов и лампы индикации

Пакет состоит из ряда упражнений, сгруппированных по 6-ти уровням сложности (см. Рис. 1.1). Упражнения описывают применение контроллеров в различных сферах деятельности. Например, сортировка деталей по размерам, управление конвейером, сверление, управление светофором, параметрами сцены, устройством оповещения в ресторане и т.д. Первые вводные упражнения содержат подробные комментарии по работе с пакетом, а также основные инструкции и приемы, применяемые при программировании.

Главное достоинство пакета – **3D графическая имитация**, которая моделирует работу оборудования в соответствии с созданной программой в режиме off-line. Кроме того, структура экрана обучения включает в себя панель управления, посредством которой осуществляется имитация управления переключателями, таблицу состояния входов-выходов и световое табло. Все это позволяет ускорить процесс отладки программы. Начиная с 4-й лабораторной работы, в задачу студента входит самостоятельная разработка программы в строгом соответствии с заданной ситуацией. Упражнения содержат подсказки и советы по подтверждению соответствия созданной программы заданным условиям.

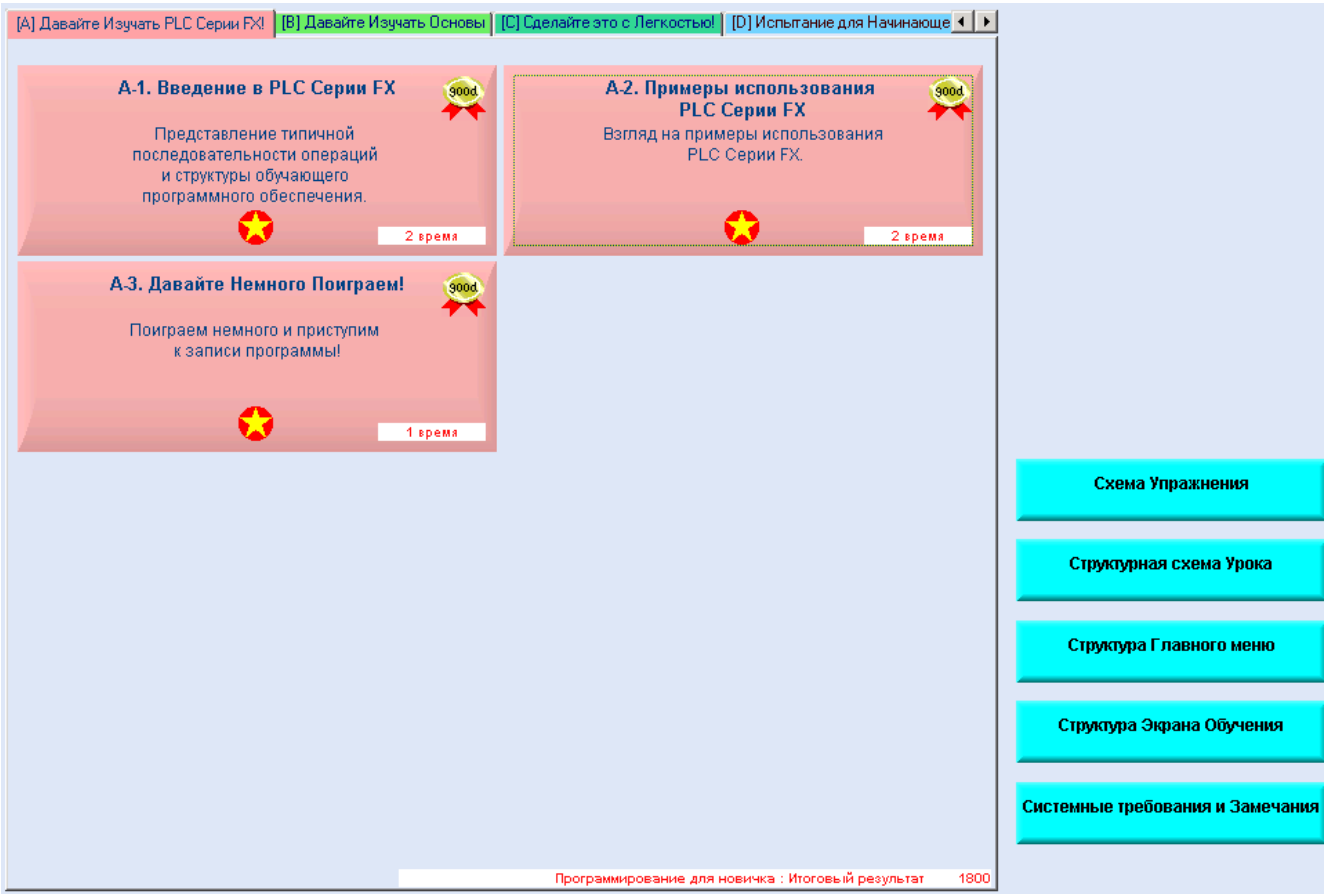



Рис. 1.1. FX Trainer. Структура Главного меню

1. **Упражнение А-1 – Представление типичной последовательности операций и структуры обучающего программного обеспечения.**

Для первоначального ознакомления со структурой обучающего пакета FX Trainer достаточно изучить структуру **Главного меню**, и затем **схему Упражнения, структурную схему Урока** и **структуру Экрана обучения**. Доступ к их подробным описаниям осуществляется нажатием соответствующих кнопок **Главного меню** (см. Рис. 1.1). Подробные пошаговые инструкции по работе с пакетом, записи программы и тестирования ее работы посредством 3D **графической имитации** приводятся в **Окне навигатора** (см. Рис. 1.2).

**Задание**

1.1Изучить структуру обучающей программы FX Trainer.

1.2Разобраться с последовательностью функционирования СУ автоматическим открытием-закрытием двери, приведенной в упражнении А-1.

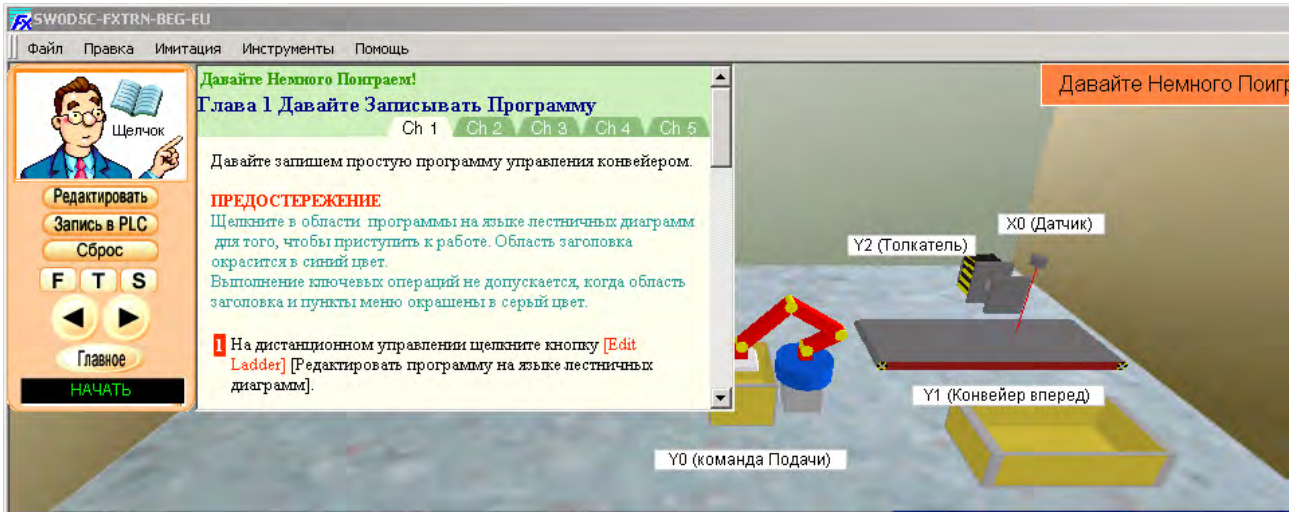



Рис. 1.2. FX Trainer. Окно навигатора

## 2. Упражнение А-2 – Примеры использования ПЛК.

**Задание**

Рассмотреть примеры использования ПЛК в различных отраслях народного хозяйства, приведенные в упражнении А-2. Более подробные сведения о назначении ПЛК, их структуре и классификации приводятся в разделе 1 «Общие сведения. Введение в ПЛК» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

## 3. Упражнение А-3 – Управление конвейером.

На рисунке 1.3 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** задает для робота с выхода *Y0* управляющий сигнал подачи детали на конвейер. Тумблер *SW1* (вход *X24* контроллера) управляет пуском-остановом конвейера (выход *Y1*) в положениях *ON/OFF* соответственно. Датчик (вход *X0*) фиксирует прохождение детали и останавливает конвейер, чтобы деталь не упала на пол. Кнопка *PB2* (вход *X21* контроллера) задает управляющий сигнал с выхода *Y2* контроллера «Столкнуть деталь в поддон» для толкателя.

Для реализации поставленной задачи управления конвейером предлагается система управления, структурная схема которой показана на рисунке 1.4.

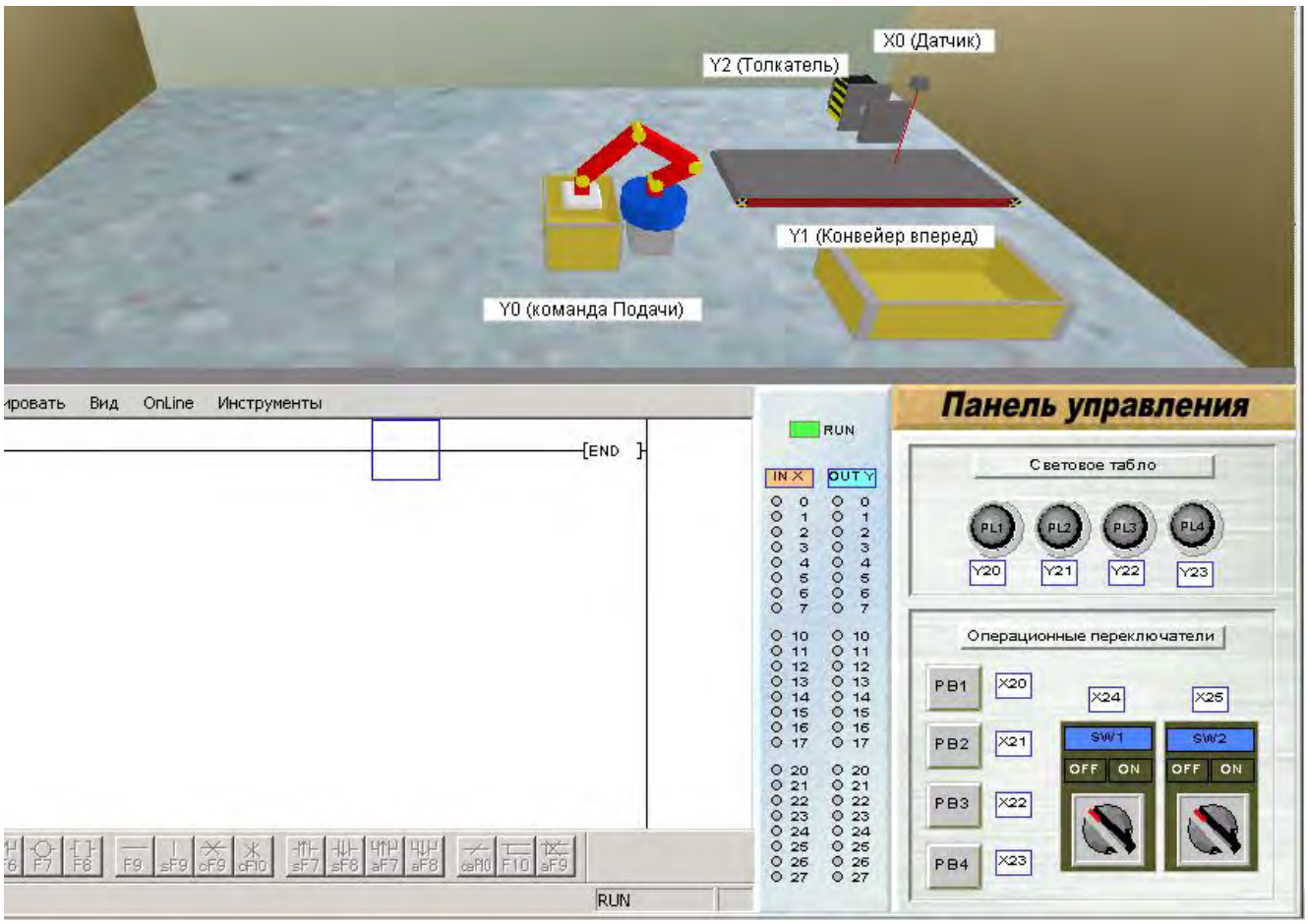



Рис. 1.3. Панель управления и виртуальное оборудование к упражнению А-3

Основным управляющим элементом системы является ПЛК, который по сигналам датчика реализует управление технологическим оборудованием. Выходные сигналы датчика поступают на входы контроллера. Обработка сигналов датчика осуществляется программно.

Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера. Срабатывание пневмоцилиндра толкателя обеспечивается подачей напряжения с выхода контроллера на электромагнитный клапан управления пневмоцилиндром. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 1.5, 1.6.

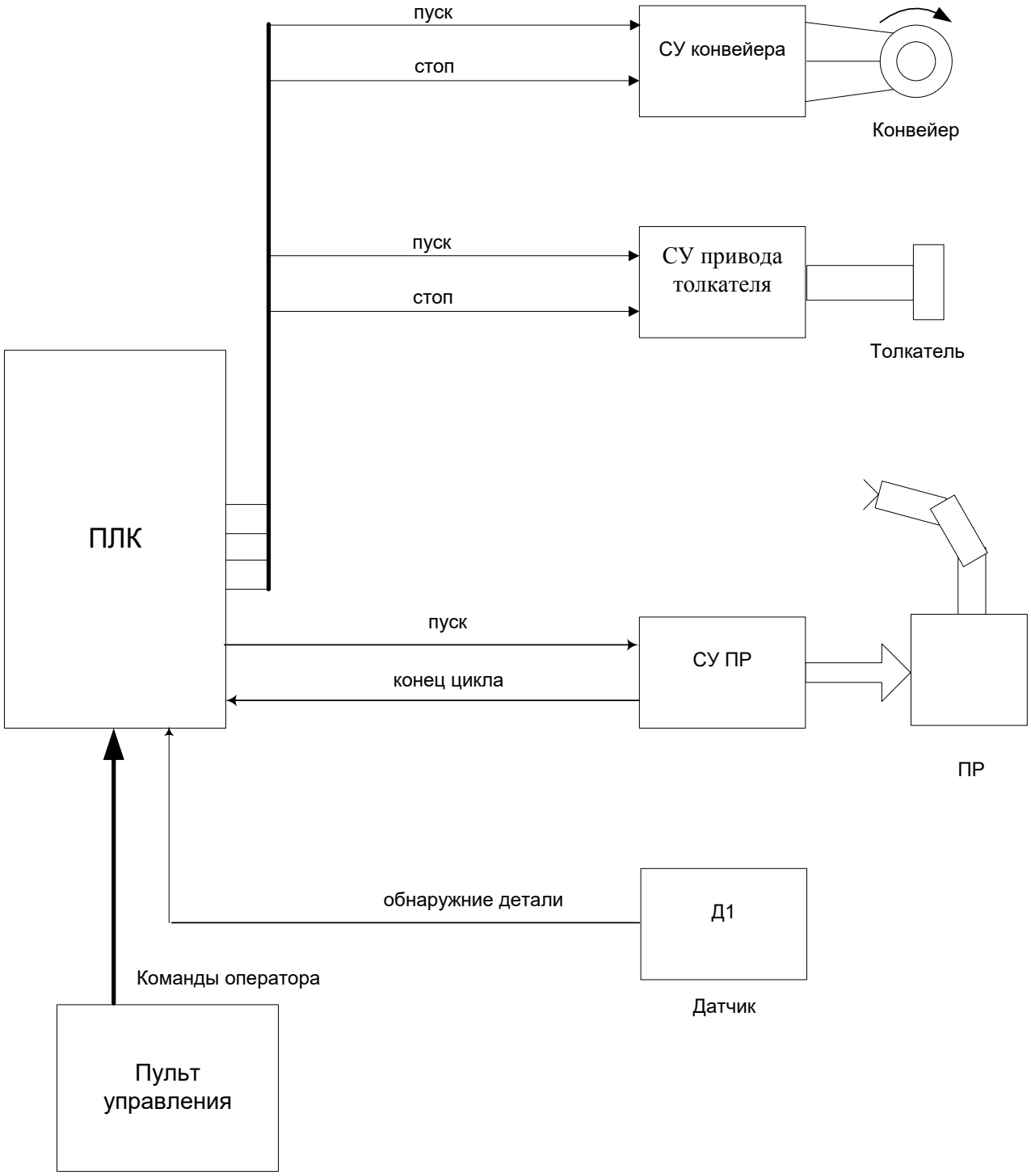



Рис. 1.4. Структурная схема СУ к упражнению А-3

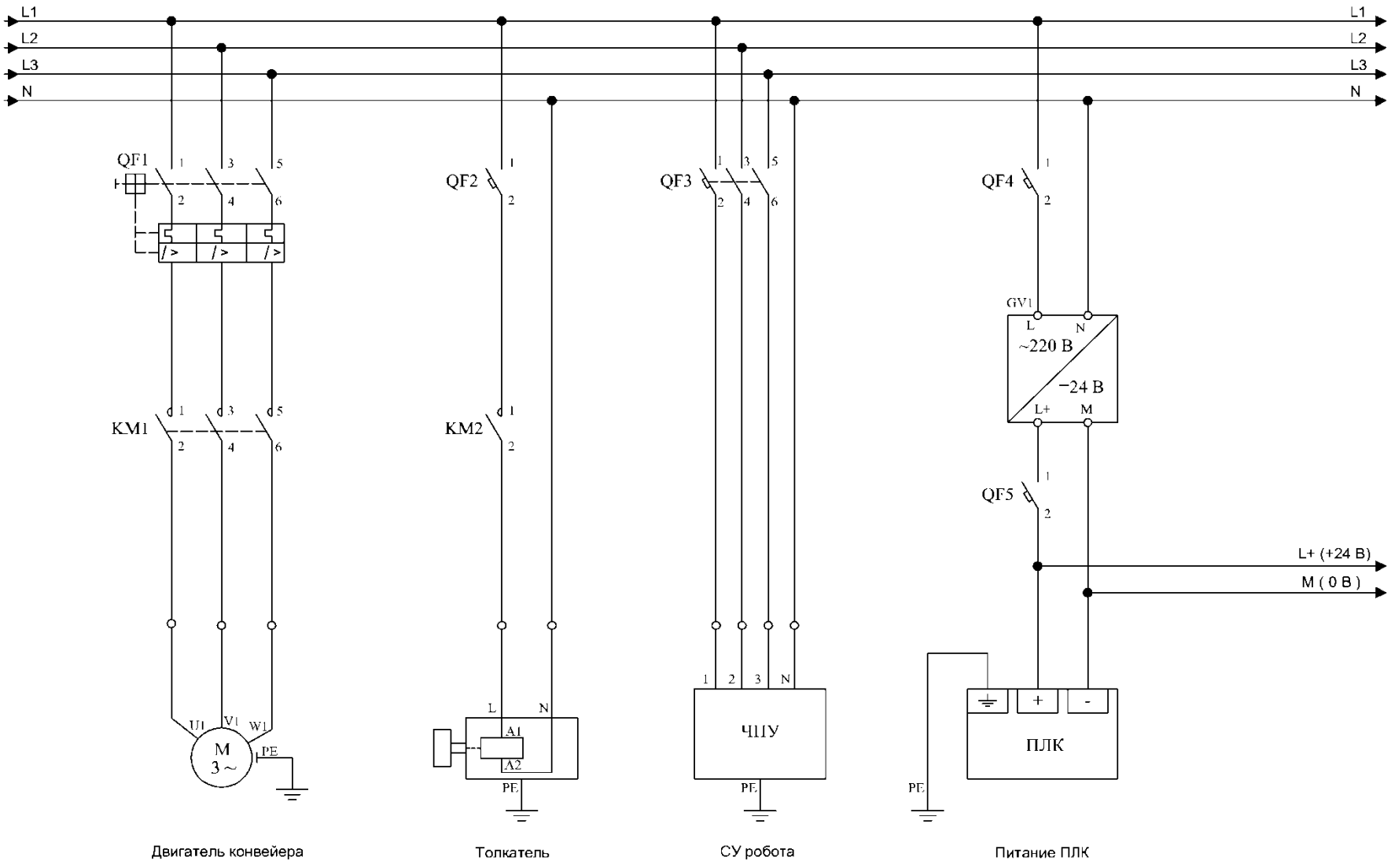



Рис. 1.5. Схема подключения оборудования (упражнение А-3)

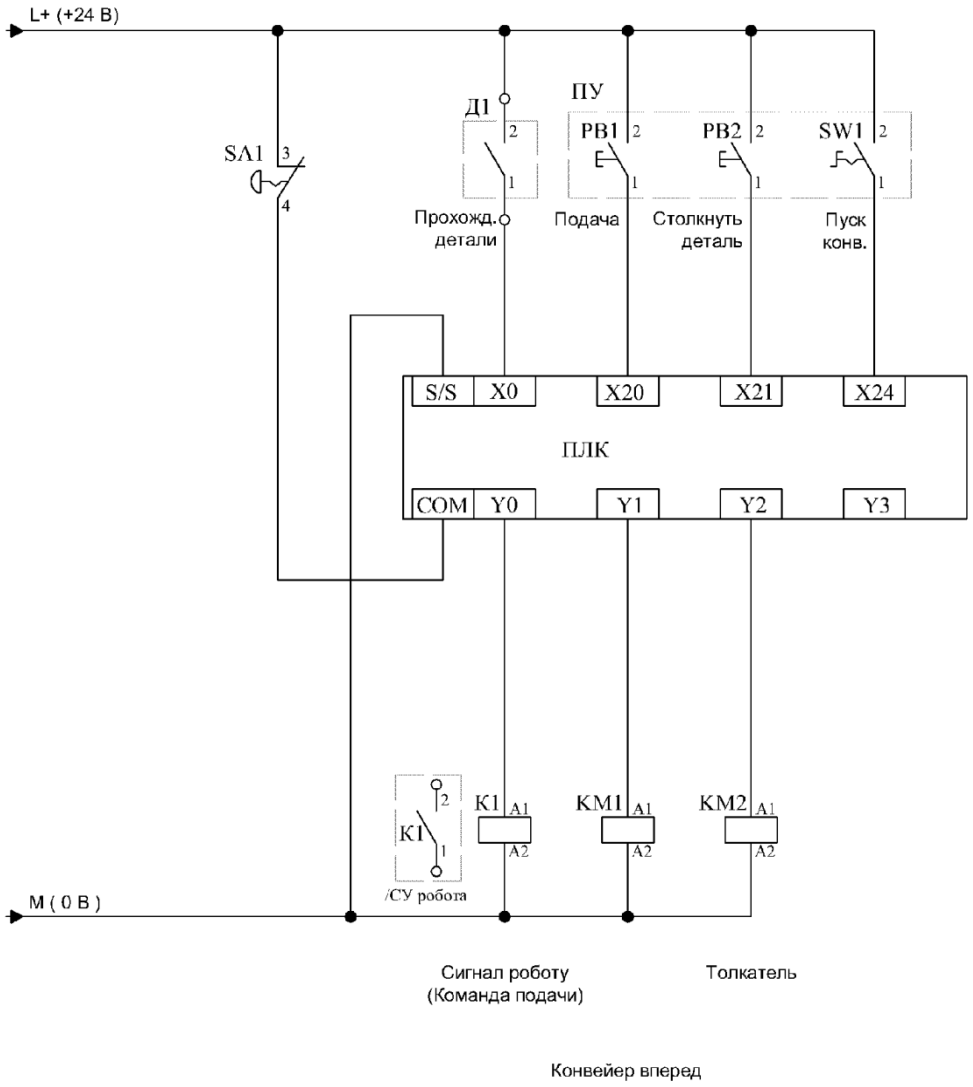



Рис. 1.6. Схема подключения входов-выходов ПЛК (упражнение А-3)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF5 – автоматические выключатели
KM1 – контактор для управления двигателем
KM2 – контактор для управления толкателем
K1 – реле для подачи управляющего сигнала роботу
Д1 – датчик положения детали
PB1, PB2 – кнопки на панели управления
SW1 – тумблер на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Задание**. Т.к. упражнение носит обучающий характер, подробные инструкции по записи программы приведены в **Окне навигатора**, т.е. в задачу студента входит не самостоятельная разработка программы, а лишь обучение ее написанию, имитация записи программы в память ПЛК и проверка ее соответствия заданным условиям. Теоретические сведения по операторам, которые используются при написании программы, приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 1 «Основные команды» Первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

3.1. Следуя приведенным в упражнении А-3 инструкциям, набрать с помощью мыши и клавиатуры управляющую программу для реализации поставленной задачи управления конвейером с учетом следующих условий:

a) Когда на панели управления нажата кнопка *PB1 (X20)*, деталь подается на конвейер.

b) Когда тумблер *SW1 (X24)* на панели управления установлен в *ON*, конвейер выполняет перемещение детали вперед.

c) Деталь останавливается у датчика *X0* и при нажатии кнопки *PB2 (X21)* сталкивается на поддон.

3.2. В соответствии с инструкциями, приведенными в **Окне навигатора**, выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

3.3. Составить схему алгоритма управляющей программы.

**Контрольные вопросы**

1. Что такое программируемое логическое управление?
2. Приведите примеры использования контроллеров в различных отраслях промышленности и в повседневной жизни.
3. Перечислите команды, которые были использованы при написании программы управления конвейером в заданииА-3.
4. Какие 3 уровня совместимости инструментальных систем определяет международная Ассоциация PLCopen и что они предполагают?
5. Какие языки программирования контроллеров Вы знаете?
6. Поясните назначение элементов на схеме подключения ПЛК.

# ЛАБОРАТОРНАЯ РАБОТА №2

## Изучение основных приемов, применяемых при программировании контроллеров

**Цель работы**

1. Ознакомиться с операторами, применяемыми при программировании контроллера: входы, выходы, логическое И и логическое ИЛИ.
2. Изучить программу выхода с защелкой и SET/RST.
3. Изучить программу контроля приоритета.
4. Ознакомиться с инструкциями для обнаружения переднего/заднего фронта импульса.

### 1. Упражнение B-1 – Изучение программ входов и выходов.

На рисунке 2.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** в состоянии *ON* включает лампу *Функционирование* (выход *Y0* контроллера). При переходе кнопки *PB1* в состояние *OFF* лампа *Функционирование (Y0)* гаснет и загорается лампа *STOP* (выход *Y1* контроллера). Лампа *Ошибка* (выход *Y2* контроллера) горит при соблюдении следующих условий:

1. Тумблер *SW1* (вход *X24* контроллера) переключен в состояние *ON*.
2. На **Панели управления** нажата кнопка *PB2 (X21)* или *PB3 (X22)*.

**Задание**

Теоретические сведения по операторам, которые используются при написании программы, приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 1 «Основные команды» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

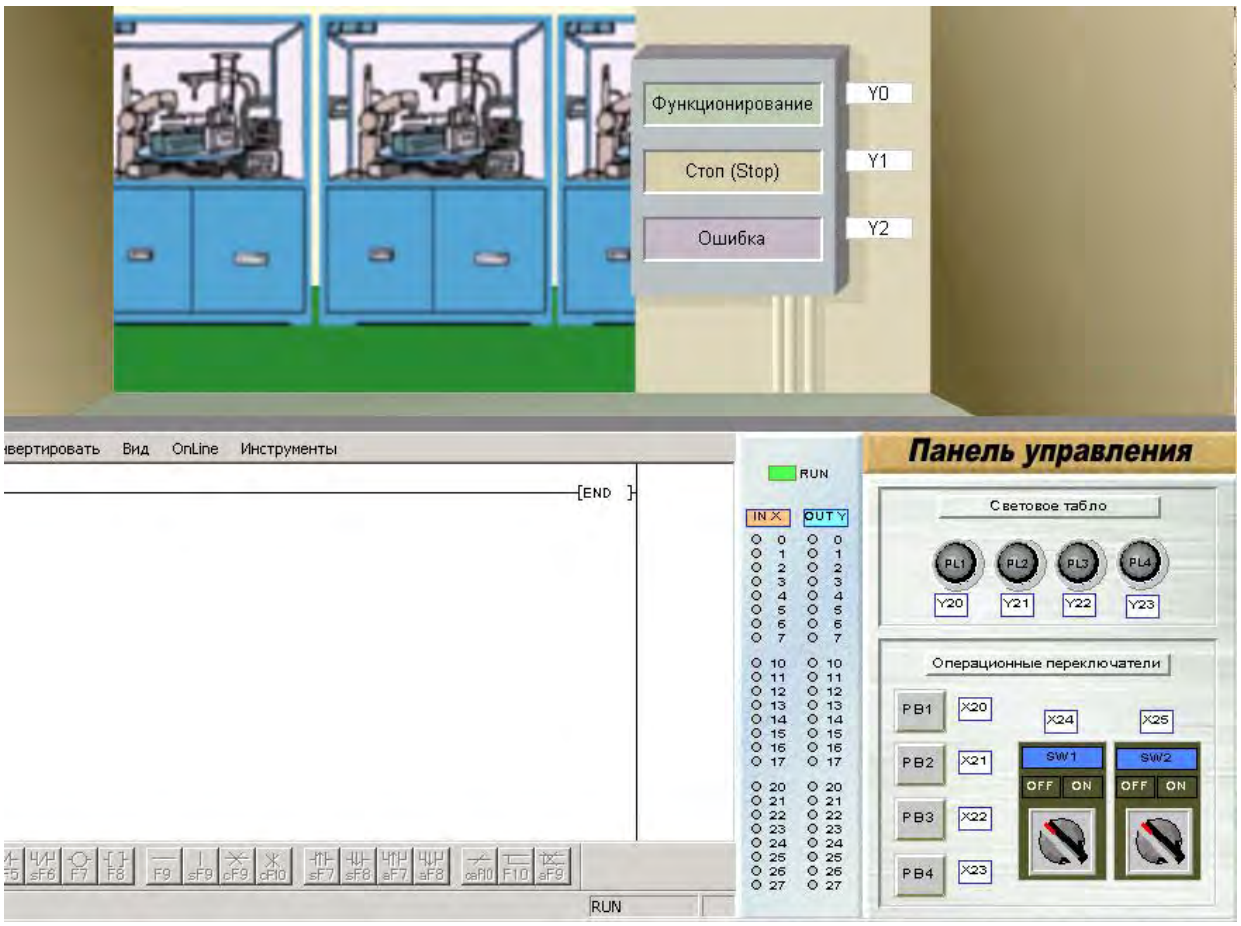



Рис. 2.1. Панель управления и виртуальное оборудование к упражнениям B-1 и B-2

1.1. Следуя приведенным в упражнении B-1 инструкциям, набрать управляющую программу для реализации поставленной задачи включения-выключения ламп с учетом следующих условий:

a) Когда на панели управления *кнопка PB1 (X20)* находится в состоянии *OFF*, горит лампа *STOP (Y1)*. Лампа *STOP (Y1)* гаснет при переходе *PB1 (X20)* в состояние *ON.*

b) Когда *PB1 (X20)* на панели управления нажата, горит лампа *Функционирование (Y0).*

c) Если тумблер *SW1 (X24)* переключен в *ON* и на панели управления нажата кнопка *PB2 (X21)*, то горит лампа *Ошибка (Y2)*.

d) Если тумблер *SW1 (X24)* переключен в *ON* и на панели управления нажата кнопка *PB3 (X22)*, то горит лампа *Ошибка (Y2).*

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.**

1.3. Разработать схему алгоритма управляющей программы.

2. **Упражнение B-2 – Программа выхода с защелкой и инструкции SET/RST.**

В упражнении B-2 предложено то же виртуальное оборудование, что и в упражнении B-1 (см. Рис. 2.1), однако управление им с помощью тех же кнопок и тумблера осуществляется иначе. Т.к. целью упражнения является изучение не только методов удержания состояния выходов контроллера, но и приемов контроля приоритета одних входов-выходов контроллера над другими, оно состоит из трех небольших программ, которые дают наглядный пример по применению этих приемов.

При нажатии на **Панели управления** кнопки *PB1* (вход *X20* контроллера) загорается лампа *Функционирование* (выход *Y0* контроллера), причем лампа продолжает гореть даже когда кнопка *PB1 (X20)* отпущена. Тумблер *SW1 (*вход *X24)* в положении *ON* выключает лампу *Функционирование (Y0)*.

**Задание**

По приведенным в **Окне навигатора** трем небольшим программам с подробными описаниями к каждой необходимо разобраться с методами удержания состояния выходов контроллера, изучить приемы контроля приоритета одних входов-выходов контроллера над другими.

Теоретические сведения по операторам, которые используются при написании программ, приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 1 «Основные команды» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

2.1. Программа удержания состояния выхода контроллера «с защелкой», контроль приоритета тумблера над кнопкой.

Следуя приведенным на вкладке Ch1 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи включения-выключения лампы *Функционирование (Y0)* с учетом следующих условий:

a) Когда кнопка *PB1 (X20)* на **Панели управления** нажата, горит лампа *Функционирование (Y0).*
b) Даже если кнопка *PB1(X20)* на **Панели управления** отпущена, лампа продолжает гореть. Состояние выхода удерживается в *ON* контактом лампы *Функционирование (Y0).*
c) Если тумблер *SW1(X24)* установлен в *ON,* лампа не может быть зажжена независимо от того, в каком состоянии находится кнопка *PB1(X20)* - *ON* или *OFF*. Тумблер *SW1(X24)* имеет приоритет по отношению к кнопке *PB1 (X20).*

2.2. Выполнить проверку соответствия набранной программы заданным условиям посредством 3D- **графической имитации**.

2.3. Программа удержания состояния выхода контроллера «с защелкой», контроль приоритета кнопки над тумблером.

Следуя приведенным на вкладке Ch3 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи включения-выключения лампы *Функционирование (Y0)* с учетом следующих условий:

a) Лампа *Функционирование (Y0)* включается по нажатию кнопки *PB1(X20)* и останется зажженной даже если кнопка *PB1(X20)* отпущена. Если тумблер *SW1(X24)* переключен в *ON*, лампа погаснет. Однако, если одновременно и кнопка *PB1(X20),* и тумблер *SW1 (X24)* находятся в состоянии *ON*, лампа *Функционирование (Y0)* будет гореть. Кнопка *PB1(X20)* имеет приоритет по отношению к тумблеру *SW1(X24).*

2.4. Выполнить проверку соответствия набранной программы заданным условиям посредством 3D- **графической имитации**.

2.5. Программа удержания состояния выхода контроллера с использованием инструкций SET/RST.

Следуя приведенным на вкладке Ch4 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи включения-выключения лампы *Функционирование (Y0)* с учетом следующих условий:

a) Когда кнопка *PB1 (X20)* на **Панели управления** нажата, горит лампа *Функционирование (Y0).*
b) Даже если кнопка *PB1 (X20)* на **Панели управления** отпущена, лампа продолжает гореть. Состояние выхода лампы *Функционирование (Y0)* удерживается в состоянии *ON* инструкцией SET.
c) Когда на **Панели управления** нажата кнопка *PB2 (X21),* лампа *Функционирование (Y0)* восстанавливает начальное состояние и гаснет.

2.6. Выполнить проверку соответствия набранной программы заданным условиям посредством 3D- **графической имитации**.

3. **Упражнение B-3 – Программа контроля приоритета.**

На рисунке 2.2 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Тумблер *SW1* (вход *X24* контроллера) в положении *ON* включает лампу *Красного цвета (Y0)* светофора при условии, что не горит лампа *Зеленого цвета (Y1).* Тумблер *SW2* (вход *X25* контроллера) в положении *ON* включает лампу *Зеленого цвета (Y1)* при условии, что не горит лампа *Красного цвета (Y0)* (тумблер *SW1 (X24)* переключен в положение *OFF*).

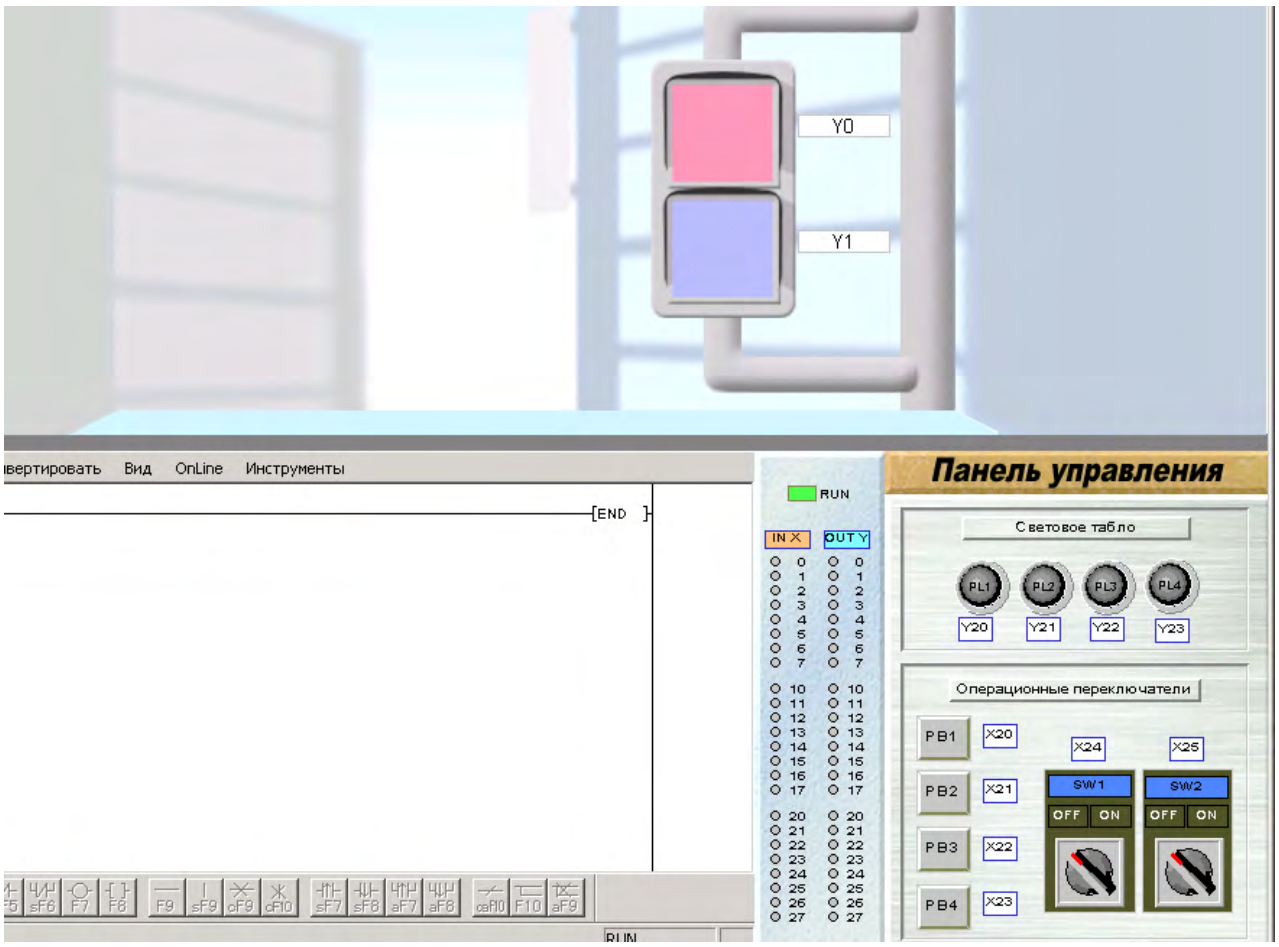



Рис. 2.2. Панель управления и виртуальное оборудование к упражнению B-3

**Задание**

По приведенным в **Окне навигатора** двум небольшим программам с подробными описаниями к каждой изучить приемы контроля приоритета одних входов-выходов контроллера над другими.

Теоретические сведения по операторам, которые используются при написании программ, приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 1 «Основные команды» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

3.1. Программа блокировки

Следуя приведенным на вкладке Ch1 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи включения-выключения ламп светофора с учетом следующих условий:

a) Когда на панели управления тумблер *SW1(X24)* установлен в *ON*, на светофоре горит лампа *Красного цвета (Y0).*

b) Даже когда на панели управления тумблер *SW2(X25)* будет установлен в *ON*, лампа *Зеленого цвета* светофора *(Y1)* не загорается. В этой программе *Зеленый свет* не появляется до тех пор, пока горит *Красный свет (Y0),* так как контакт *Y0* разомкнут.

c) При изменении функционирования на обратное (первоначальная установка в *ON* тумблер *SW2 (X25)*, и только затем тумблера *SW1 (X24) )* включение *Зеленого цвета* светофора *(Y1)* имеет высший приоритет по отношению к *Красному*.

3.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

3.3. Программа контроля приоритета

Следуя приведенным на вкладке Ch4 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи включения/выключения ламп светофора с учетом следующих условий:

a) Когда на **Панели управления** нажата кнопка *PB1 (X20),* горит сигнал *Красного цвета (Y0).*
b) Когда кнопка *PB1 (X20)* на **Панели управления** отпущена, сигнал продолжает гореть. Состояние выхода для сигнала *Красного цвета (Y0)* сохраняется программой защелки выхода.
c) Когда на **Панели управления** нажата кнопка *PB2 (X21),* сигнал *Красного цвета (Y0)* гаснет и загорается сигнал *Зеленого цвета (Y1).*
d) Когда кнопка *PB2 (X21)* на **Панели управления** отпущена, сигнал *Зеленого цвета (Y1)* продолжает гореть. Состояние выхода для сигнала *Зеленого цвета (Y1)* сохраняется программой защелки выхода.
e) Функционирование сигнала *Зеленого цвета (Y1),* который был включен в *ON* «последним», имеет более высокий приоритет.

3.4. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

4. **Упражнение B-4 – Обнаружение переднего или заднего фронта импульса.**

На рисунке 2.3 показана **Панель управления** с индикаторами состояния входов/выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Тумблер *SW1* (вход *X24* контроллера) управляет пуском/остановом конвейера (выход *Y1*) в положениях *ON/OFF* соответственно. При движении конвейера вперед горит лампа *Зеленого цвета (Y6)*. Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** задает для робота с выхода *Y0* управляющий сигнал подачи детали на конвейер. Кнопка *PB2* (вход *X21* контроллера) останавливает конвейер (при этом гаснет лампа Зеленого цвета (*Y6*)) и блокирует команду подачи для робота (*Y0*).

Для реализации поставленной задачи управления конвейером предлагается система управления, структурная схема которой показана на рисунке 2.4.

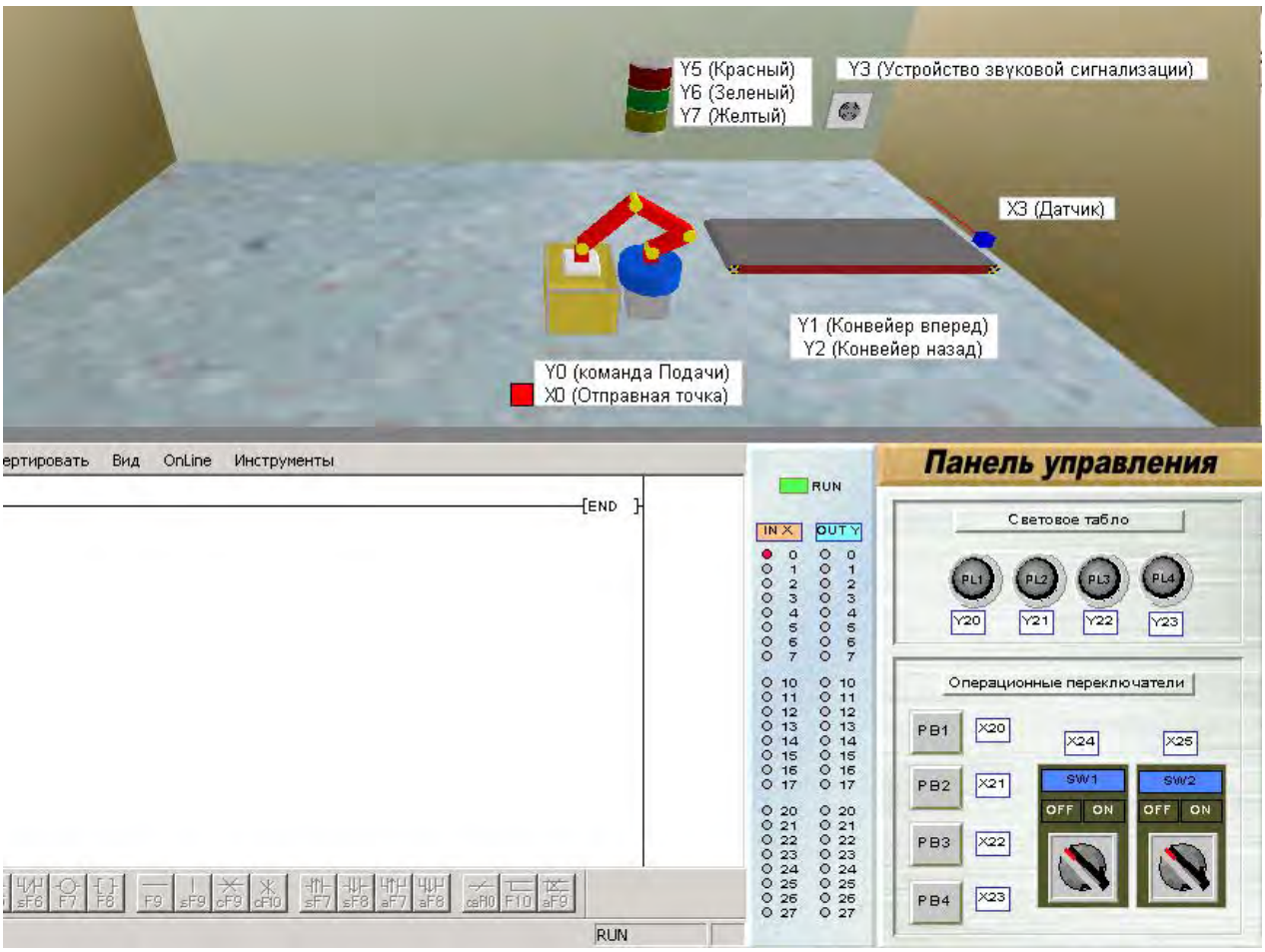



Рис. 2.3. Панель управления и виртуальное оборудование к упражнению В-4

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление технологическим оборудованием. Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 2.5, 2.6

Целью упражнения является изучение инструкций PLS/PLF, LDP/LDF, а также определение ситуаций, где наиболее предпочтительно использовать те или иные инструкции. В связи с этим оно состоит из трех небольших программ, которые дают наглядный пример по применению этих инструкций.

**Задание**

По приведенным в **Окне навигатора** трем небольшим программам с подробными описаниями к каждой необходимо разобраться с инструкциями определения переднего/заднего фронта импульса.

Теоретические сведения по операторам управления по фронтам входных сигналов LDP/LDF приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 1 «Основные команды» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами», по операторам генерации одиночных импульсов по фронтам входных сигналов PLF/PLS – в главе 5 «Программирование одиночных импульсов. Команды (PLF) и (PLS)», по программированию внутреннего реле – в главе 2 «Программирование внутреннего реле».

4.1. Инструкции PLS/PLF

Следуя приведенным на вкладках Ch1-Ch4 **Окна навигатора** инструкциям, набрать управляющую программу для реализации поставленной задачи управления технологическим оборудованием с учетом следующих условий:

a) В момент переключения тумблера *SW1 (X24)* в состояние *ON Конвейер (Y1)* начинает движение *вперед* и загорается *Зеленая лампа (Y6)*.
b) Когда нажата кнопка *PB2 (X21),* выходы *Y1* и *Y6* переключаются в *OFF*.
c) Команда подачи (*Y0)* для робота срабатывает в момент переключения кнопки *PB1 (X20)* из состояния *ON* в состояние *OFF.*
d) Функционирование робота *Y0* прекращается при нажатии на кнопку *PB2 (X21).*

4.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

4.3. Инструкции LDP/LDF

Не изменяя функциональное назначение кнопок и тумблера на **Панели управления** реализуйте управление технологическим оборудованием при помощи инструкций LDP/LDF, следуя указаниям, приведенным на вкладках Ch5-Ch6.

4.4. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

4.5. Способ предотвращения случайных запусков оборудования

Следуя приведенным на вкладке Ch7 Окна навигатора инструкциям, набрать управляющую программу для реализации поставленной задачи предотвращения случайных запусков оборудования с учетом следующих условий:

a) Лампа *Красного цвета (Y5)* горит при условии, что тумблер *SW1 (X24)* установлен в положение *ON.* При этом лампа *Y5* гаснет во время нажатия на кнопку *PB2 (X21)* и загорается вновь при ее отпускании.
b) В момент переключения тумблера *SW1 (X24)* в состояние *ON* загорается *Зеленая лампа (Y6)*. При этом лампа *Y6* выключается при нажатии на копку *PB2* (*X21*) на **Панели управления** и не загорается вновь до тех пор, пока тумблер *SW1 (X24)* не будет переключен в положение *OFF*, а затем снова в *ON*. Этот прием предотвращает случайный запуск оборудования.

4.6. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

**Структурная схема системы управления конвейером** к упражнению В-4

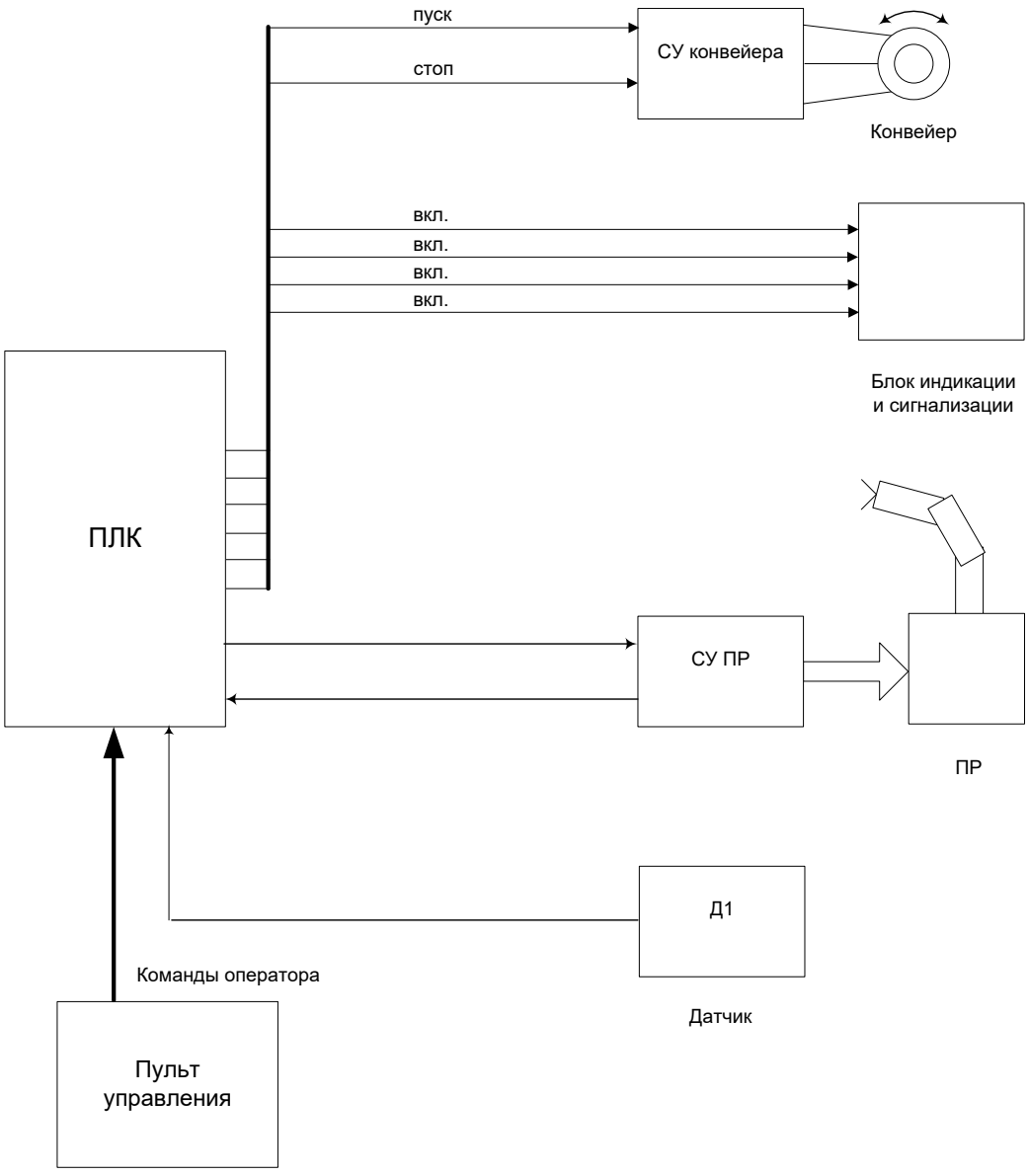



Рис. 2.4. Структурная схема СУ к упражнению В-4

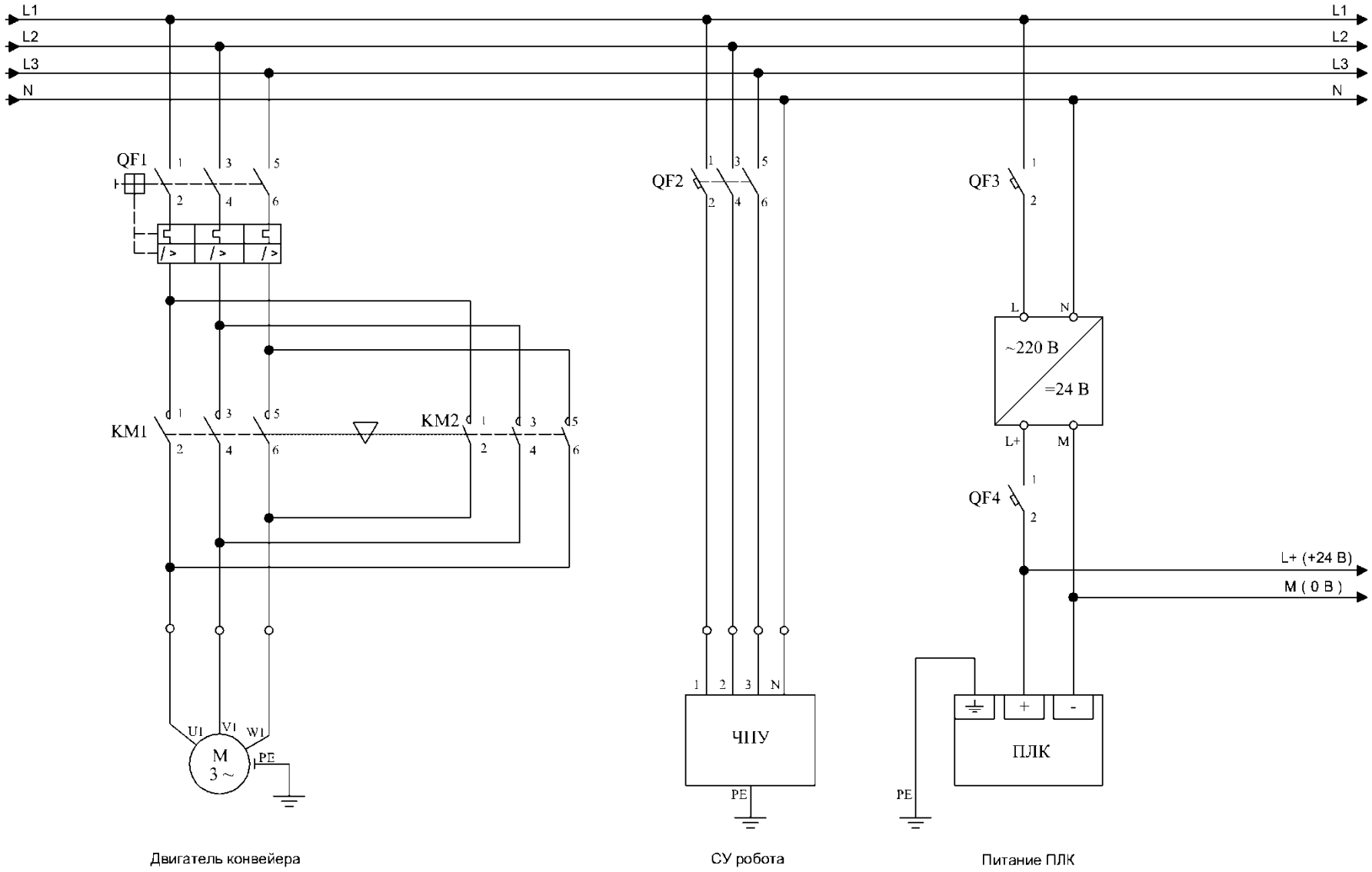



Рис. 2.5. Схема подключения оборудования (упражнение В-4)

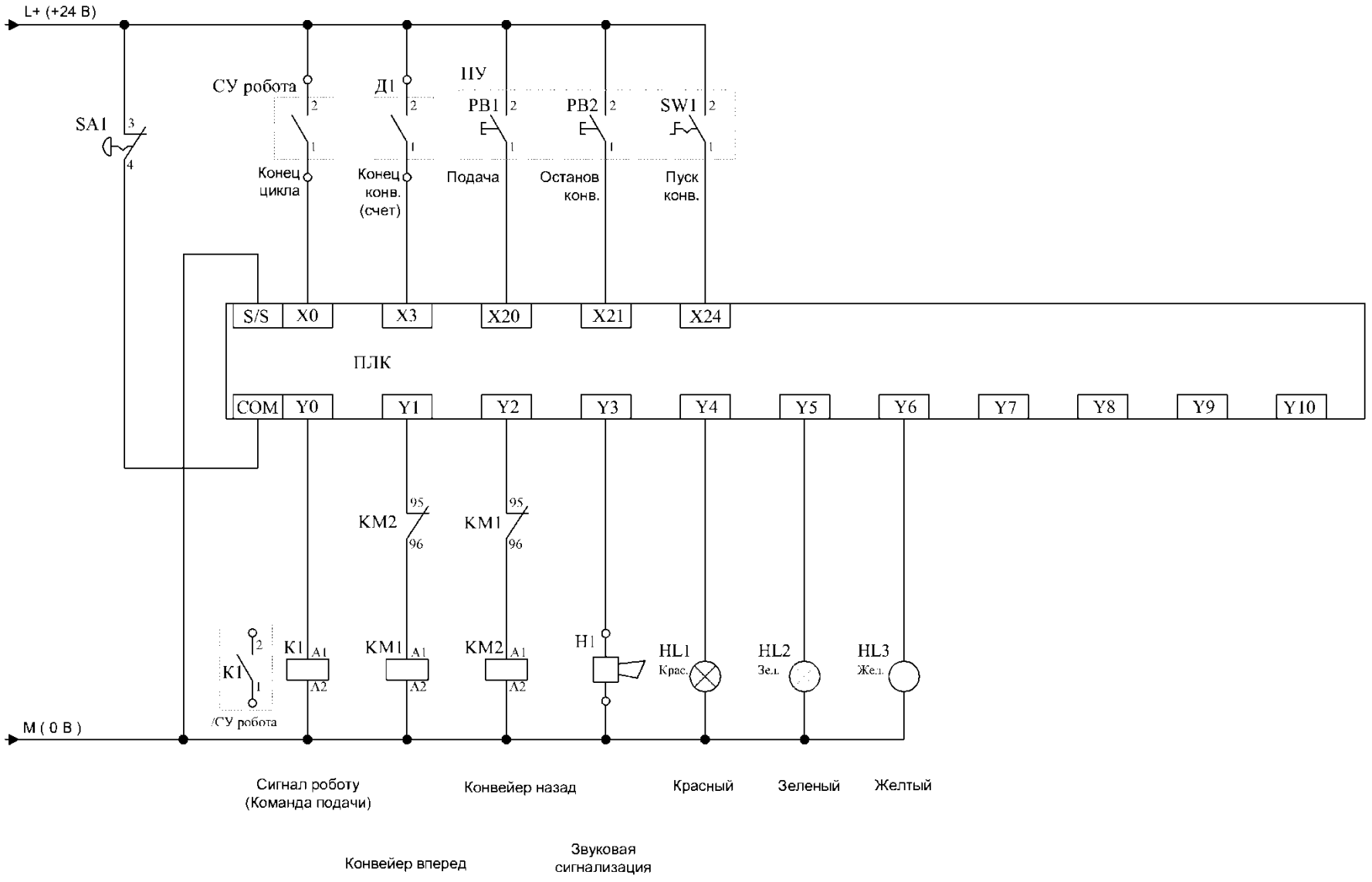



Рис. 2.6. Схема подключения входов-выходов ПЛК (упражнение В-4)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF4 – автоматические выключатели
KM1 – контактор для управления двигателем (вперед)
KM2 – контактор для управления двигателем (реверс)
K1 – реле для подачи управляющего сигнала роботу
Д1 – датчик положения детали
H1 – Звуковая сигнализация
HL1, HL2, HL3 – световая сигнализация (красная, зеленая, желтая соответственно)
PB1, PB2 – кнопки на панели управления
SW1 – тумблер на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Как работают нормально разомкнутые и нормально замкнутые контакты?
2. Объясните логику работы «Последовательного И» и «Параллельного ИЛИ».
3. Для чего используются внутренние реле ПЛК?
4. Какие методы удержания состояния выхода Вы знаете?
5. Приведите примеры использования инструкций с высоким приоритетом.
6. В каких случаях используются инструкции PLS и PLF?
7. В чем принципиальное отличие между инструкциями PLS/PLF и LDP/LDF?
8. Внесите необходимые изменения в первую или вторую программу упражнения В-4 по управлению технологическим оборудованием, чтобы заготовка при движении по конвейеру не падала на пол, а автоматически останавливалась по сигналу датчика (вход *X3* контроллера).

# ЛАБОРАТОРНАЯ РАБОТА №3

## Основные функции таймера. Методы управления на основе счетчиков

**Цель работы**

1. Изучить основные функции и виды таймеров, приемы их программной реализации.
2. Ознакомиться с методами управления на основе счетчиков.

### 1. Упражнение С1 – Основные функции таймера.

На рисунке 3.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** в состоянии *ON* включает таймер *T0* контроллера. Когда текущее значение таймера достигает 3с, с выхода *Y0* контроллера выдается сигнал открытия двери (*Команда Дверь вверх*). При переходе кнопки *PB1* в состояние *OFF* текущее значение таймера *T0* обнуляется. Для управления закрытием двери необходимо нажать и удерживать кнопку *PB2 (X21)* на **Панели управления** до тех пор, пока текущее значение таймера *T1* контроллера не достигнет 4с. В этот момент с выхода *Y1* контроллера подается сигнал закрытия двери (*Команда Дверь вниз*). При переходе кнопки *PB2* в состояние *OFF* текущее значение таймера *T1* обнуляется.

Для реализации поставленной задачи управления открытием-закрытием двери предлагается система управления, структурная схема которой показана на рисунке 3.2.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов и сигналам датчиков *Верхнего предела* Д1 и *Нижнего предела* Д2 по записанной в память программе изменяет состояние выходов, т.е. реализует управление открытием-закрытием двери. Дверь приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку двери.

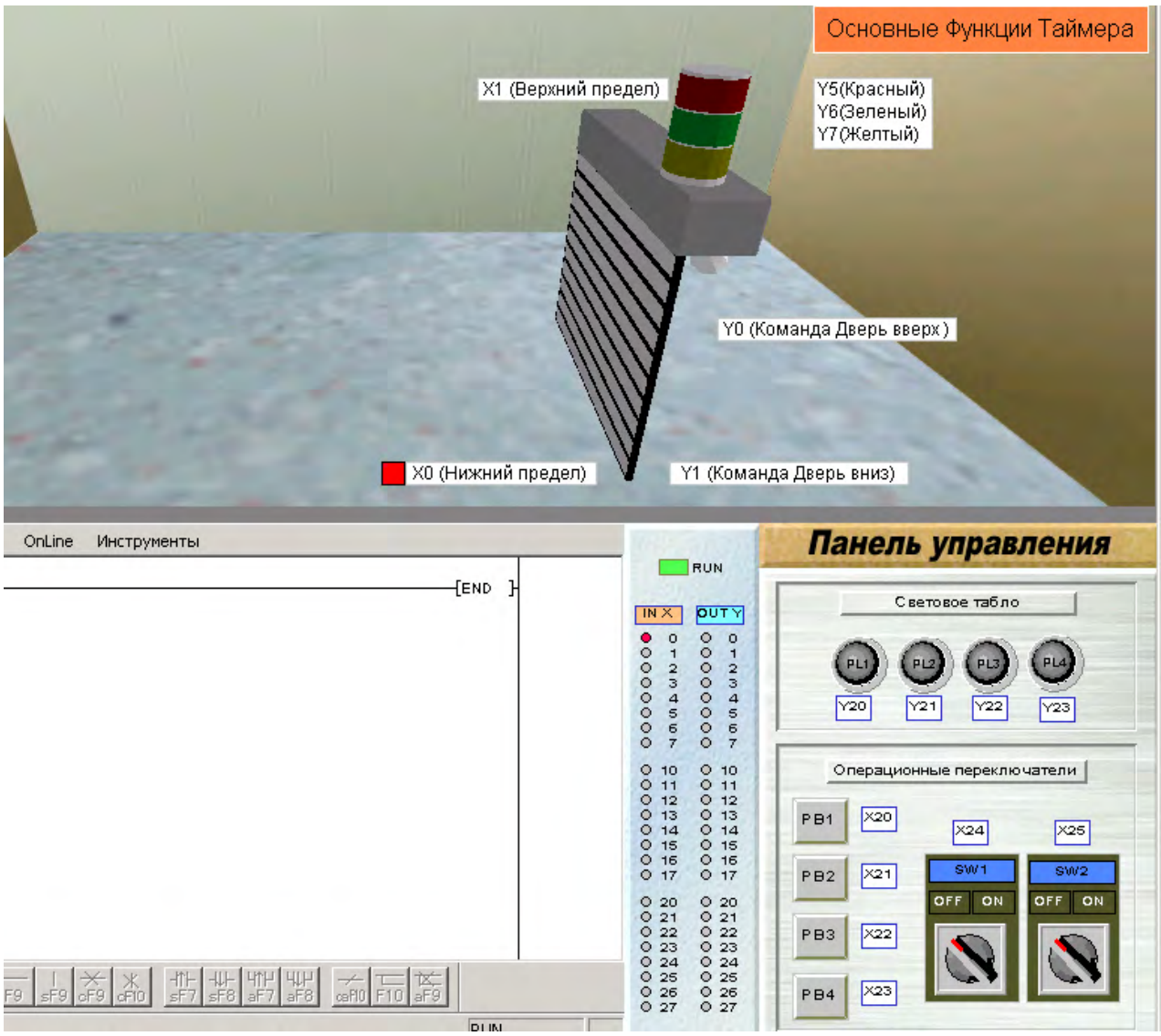



Рис. 3.1. Панель управления и виртуальное оборудование к упражнениям С-1 и С-2

Схема подключения оборудования приведена на рисунке 3.3.

**Задание**

Теоретические сведения по назначению таймеров, их классификации, инициализации и примерам использования приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 4 «Программирование таймера. Команда TIMER» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

1.1. Следуя приведенным в упражнении С-1 инструкциям, набрать управляющую программу для реализации поставленной задачи управления открытием-закрытием двери с учетом следующих условий:

а) При нажатии и удерживании на **Панели управления** кнопки *PB1 (X20)* запускается (устанавливается в состояние *ON*) программа таймера *T0*.

b) Спустя 3 секунды после того, как *T0* установлен в *ON*, запускается на выполнение *Команда Дверь вверх Y0.*

c) Если на **Панели управления** нажать кнопку *PB2 (X21),* то таймер *T1* установится в *ON*, и через 4 секунды дверь начнет закрываться.

d) В момент перехода в состояние *OFF* кнопок *PB1 (X20)* или *PB2 (X21)* текущее значение таймеров *T0* или *T1* соответственно обнуляется.

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

**Структурная схема системы управления** к упражнениям С-1 и С-2

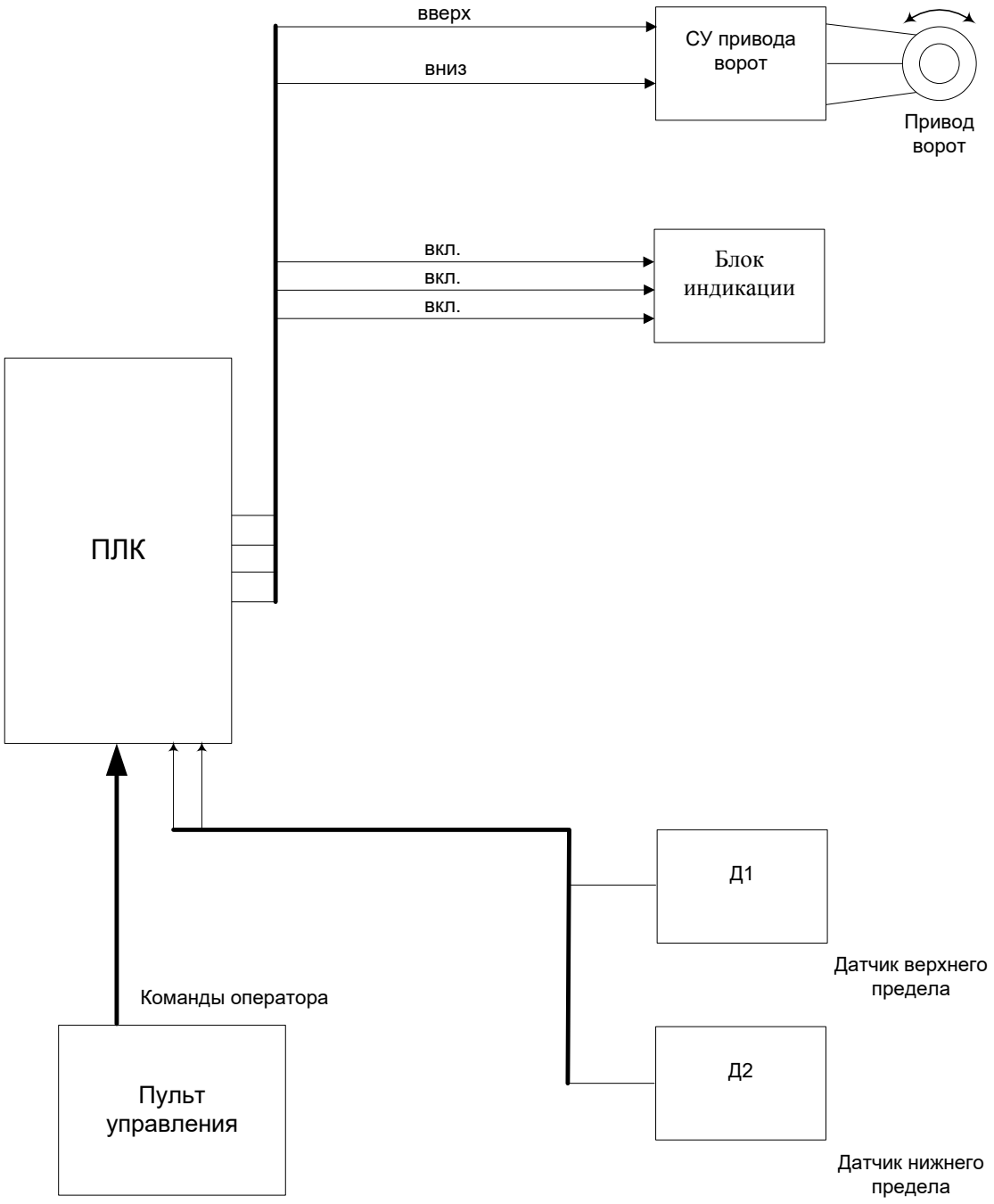



Рис. 3.2. Структурная схема СУ к упражнениям С-1 и С-2

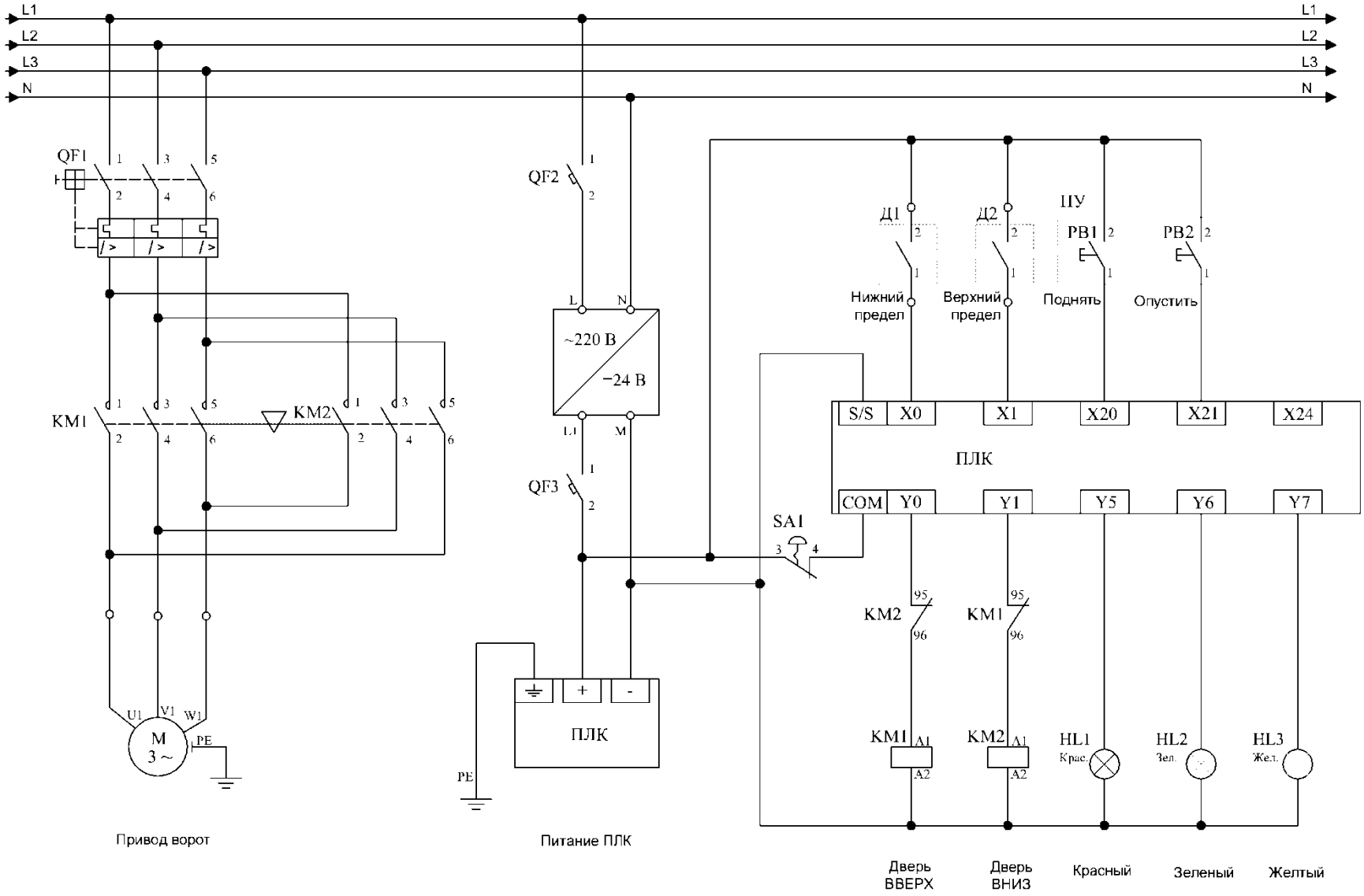



Рис. 3.3. Схема подключения оборудования (упражнения С-1 и С-2)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2, QF3 – автоматические выключатели
KM1 – контактор для управления двигателем (ворота ВВЕРХ)
KM2 – контактор для управления двигателем (ворота ВНИЗ)
Д1, Д2 – датчики положения ворот
HL1, HL2, HL3 – световая сигнализация (красная, зеленая, желтая соответственно)
PB1, PB2 – кнопки на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

## 2. Упражнение С-2 – Изучение программы таймера с задержкой выключения и таймера-одновибратора.

В упражнении С-2 предложено то же виртуальное оборудование, что и в упражнении С-1 (см. Рис. 3.1), однако управлять мы будем не открытием-закрытием двери, а включением-выключением ламп индикации, которые расположены непосредственно над дверьми. Отсюда следует, что назначение кнопок на **Панели управления** иное.

Кнопка *PB1* (вход *X20* контроллера) в состоянии *ON* включает лампу *Красного цвета* (выход *Y5* контроллера). В момент перехода кнопки *PB1* из состояния *ON* в состояние *OFF* включается таймер *T1* контроллера. Когда текущее значение таймера достигает 3с, лампа *Красного цвета (Y5)* гаснет и текущее значение таймера *T1* обнуляется. Здесь используется прием задержки выхода *Y5* в состоянии *ON* в течение заданного времени после того, как задающий вход *X20* перешел в состояние *OFF.*

В момент перехода кнопки *PB2* (вход *X21* контроллера) из состояния *ON* в состояние *OFF* загорается лампа *Зеленого цвета* (выход *Y6* контроллера) и включается таймер-одновибратор *T2* контроллера. Когда текущее значение таймера *T2* достигает 5с, лампа *Зеленого цвета (Y6)* гаснет и таймер обнуляется. Таймер-одновибратор используется для удержания выхода *Y6* в состоянии *ON* в течение точно установленного промежутка времени, после того, как задающий вход *X21* перейдет в состояние *ON*.

Структурная схема СУ для реализации поставленной задачи включения-выключения ламп индикации показана на рисунке 3.2.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление включением-выключением ламп индикации.

Схема подключения оборудования приведена на рисунке 3.3.

**Задание**

2.1. Следуя приведенным в упражнении С-2 инструкциям, набрать управляющую программу для реализации поставленной задачи управления включением-выключением ламп индикации:

a) При нажатии на **Панели управления** на кнопку *PB1 (X20)* загорается лампа *Красного цвета (Y5)*.

b) Отпустите кнопку *PB1 (X20)* на **Панели управления**. Таймер *T1* устанавливается в состояние *ON*.

c) Спустя 3 секунды лампа *Красного цвета (Y5)* гаснет.

d) При нажатии на **Панели управления** на кнопку *PB2 (X21)* загорается лампа *Зеленого цвета (Y6)*. Таймер *T2* устанавливается в состояние *ON*.

e) Спустя 5 секунд лампа *Зеленого цвета (Y6)* гаснет.

2.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

## 3. Упражнение С-3 – Программа мерцания.

На рисунке 3.4 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления светофором.

Тумблер *SW1* (вход *X24* контроллера) в положении *ON* запускает отсчет времени таймера *T3*. Когда текущее значение таймера достигает 2с, загорается сигнал *Зеленого цвета (Y1)* светофора и включается таймер *T4*. Когда текущее значение таймера *T4* достигает 4с, сигнал *Зеленого цвета (Y1)* гаснет и таймер *T3* обнуляется.


Рис. 3.4. Панель управления и виртуальное оборудование к упражнению С-3

Структурная схема СУ для реализации поставленной задачи мерцания *Зеленого сигнала* светофора показана на рисунке 3.5.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление сигналами светофора.

Схема подключения СУ приведена на рисунке 3.6.

**Задание**

3.1. Следуя приведенным в Окне навигатора инструкциям, набрать управляющую программу для реализации поставленной задачи мерцания Зеленого сигнала светофора:

a) На **Панели управления** установите в состояние *ON* тумблер *SW1 (X24)*.
b) После 2 секунд, отсчитанных таймером *T3*, загорится сигнал *Зеленого цвета (Y1)*.
c) Сигнал *Зеленого цвета (Y1)* остается зажженным в течение 4 секунд, отсчитанных *T4*.
d) Впоследствии, *Зеленый сигнал (Y1)* будет мерцать, оставаясь в состоянии *OFF* в течение 2 секунд и в состоянии *ON* в течение 4 секунд.

3.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

**Структурная схема системы управления** к упражнению С-3

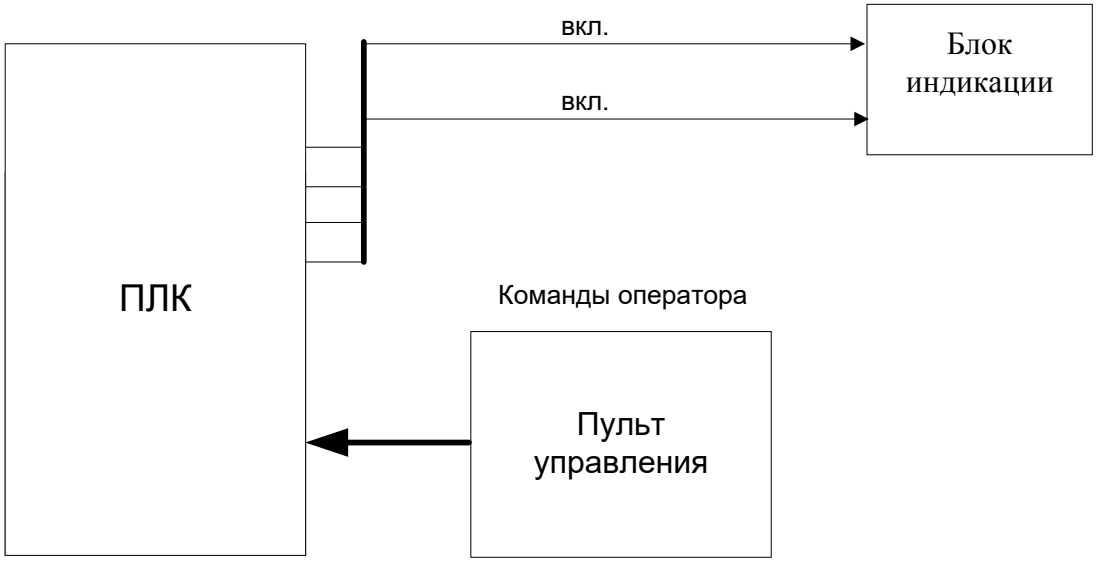



Рис. 3.5. Структурная схема СУ к упражнению С-3

**Схема подключения ПЛК** к упражнению С-3

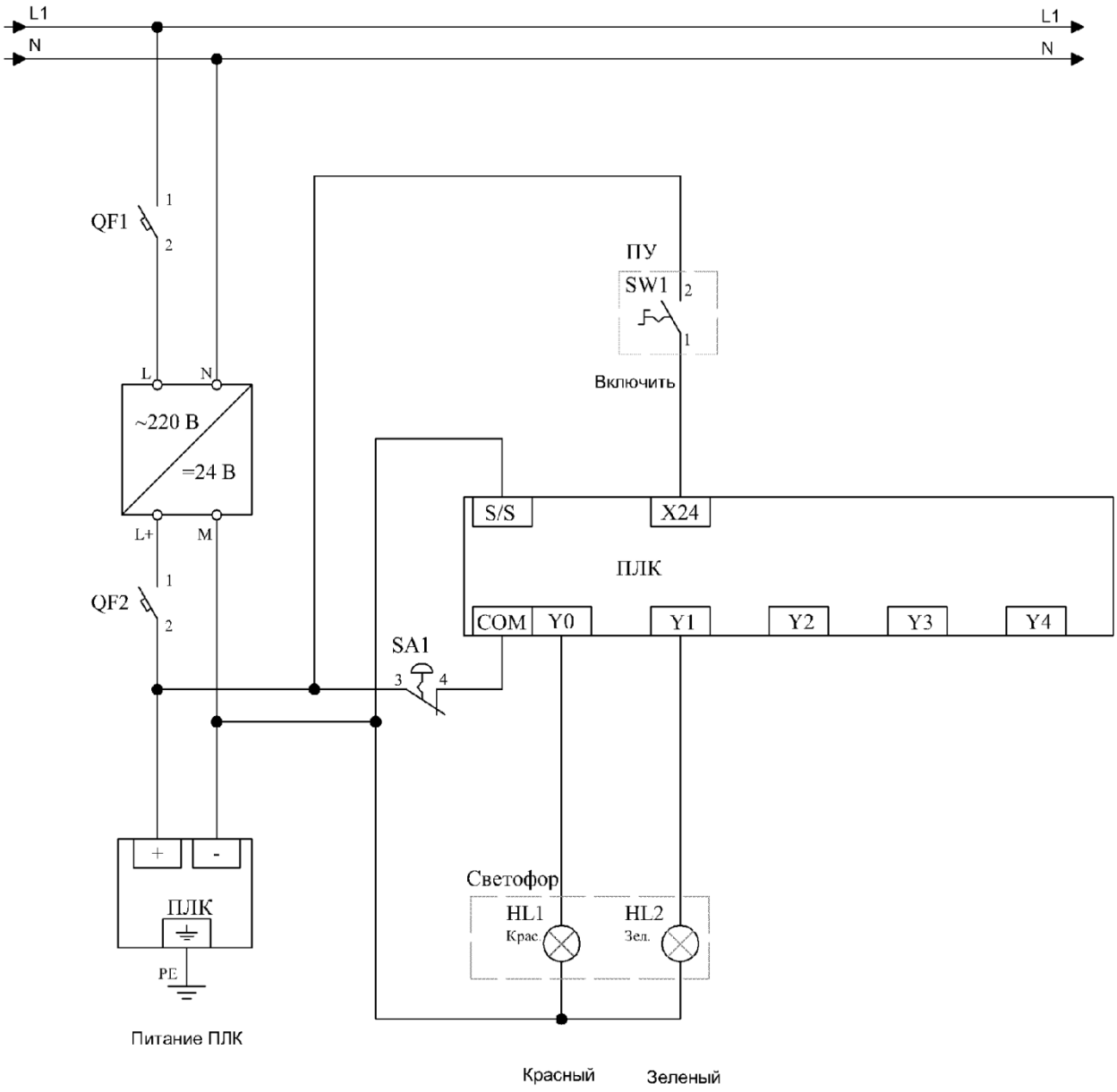



Рис. 3.6. Схема подключения СУ (упражнение С-3)

L1 – однофазный источник питания (220В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматические выключатели
HL1 – лампа красного цвета
HL2 – лампа зеленого цвета
SW1 – тумблер включения на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

4. **Упражнение С-4 – Основная программа счета.**

На рисунке 3.7 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

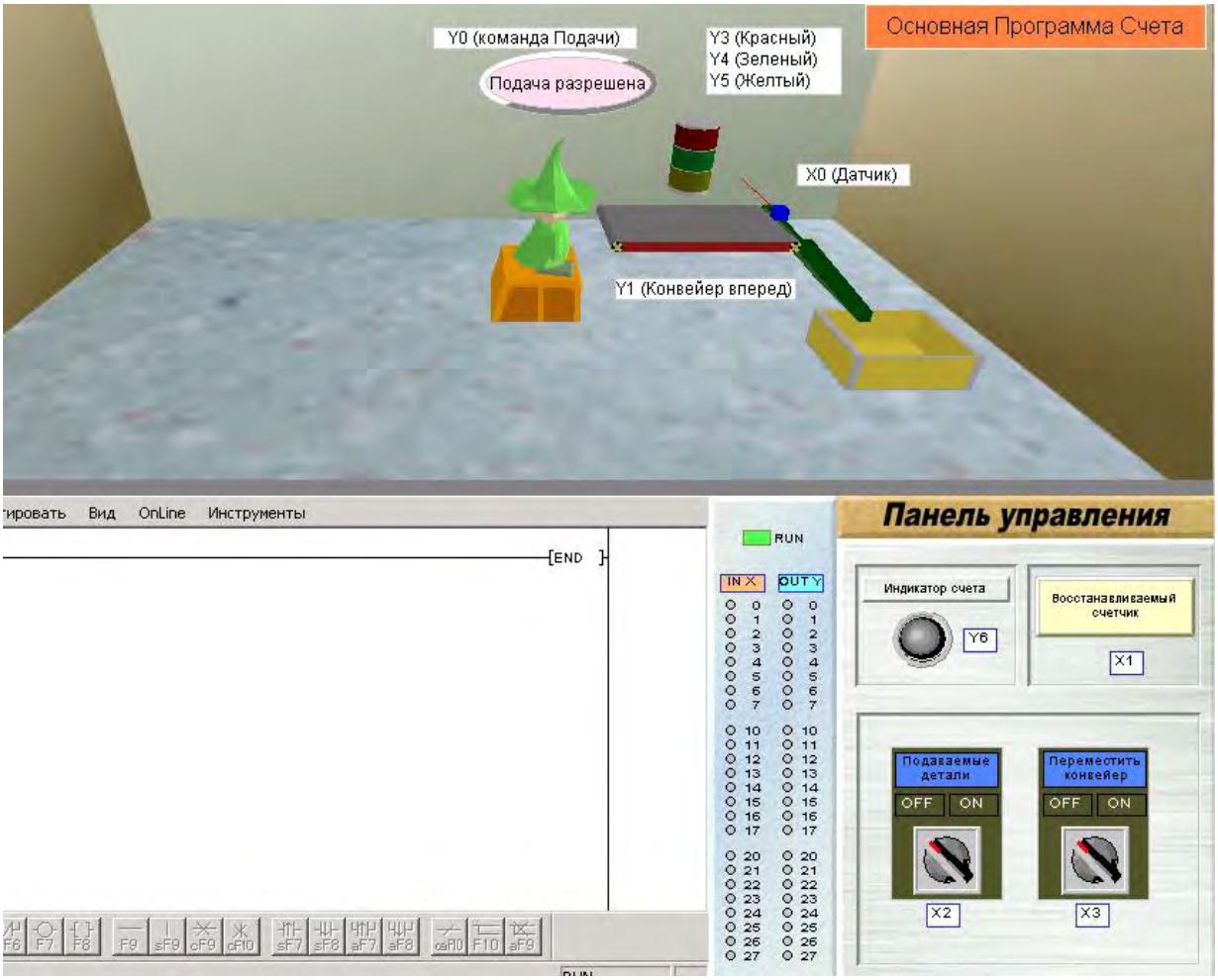



Рис. 3.7. Панель управления и виртуальное оборудование к упражнению С-4

Тумблер *X2* на **Панели управления** в состоянии *ON* включает лампу индикации *Подача разрешена (Y0)* для работника. Тумблер *X3* управляет пуском-остановом конвейера с выхода *Y1* контроллера в положениях *ON/OFF* соответственно. Датчик *X0* фиксирует прохождение детали по конвейеру. По сигналам от датчика происходит срабатывание внутреннего счетчика контроллера (*C0*). При достижении счетчиком заданного значения 10 загорается *Индикатор счета (Y6)* на **Панели управления**. Обнуление счетчика происходит при нажатии на кнопку *Обнуление счетчика (X1)*.

Для реализации поставленной задачи счета деталей предлагается система управления, структурная схема которой показана на рисунке 3.8.

Основным управляющим элементом системы является программируемый логический контроллер, который по сигналам датчика реализует управление технологическим оборудованием. Датчик Д1 регистрирует прохождение деталей, выходные сигналы датчика поступают на входы

контроллера, программно обрабатываются и инициируют работу счетчика деталей. Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера.

Схема подключения СУ приведена на рисунке 3.9.

**Задание**

Теоретические сведения по назначению счетчиков, их классификации, инициализации и примерам использования приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 3 «Программирование счетчика. Команда COUNTER» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

4.1. Следуя приведенным в Окне навигатора инструкциям, набрать управляющую программу для реализации поставленной задачи счета деталей:
   a) На **Панели управления** переключите тумблеры *Подача деталей (X2)* и *Продвижение конвейера (X3)* в состояние *ON.*
   b) Текущее значение счетчика *(C0)* должно возрастать с каждым проходом детали мимо датчика *X0*.
   c) Заданное для счетчика значение *(C0)* - *K10,* поэтому, когда текущее значение достигнет 10, загорится *Индикатор счета (Y6)* на **Панели управления**.
   d) Когда будет нажата кнопка *Обнуление счетчика (X1),* текущее значение счетчика *(C0)* установится в 0.

4.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.**

**Структурная схема системы управления** к упражнению С-4

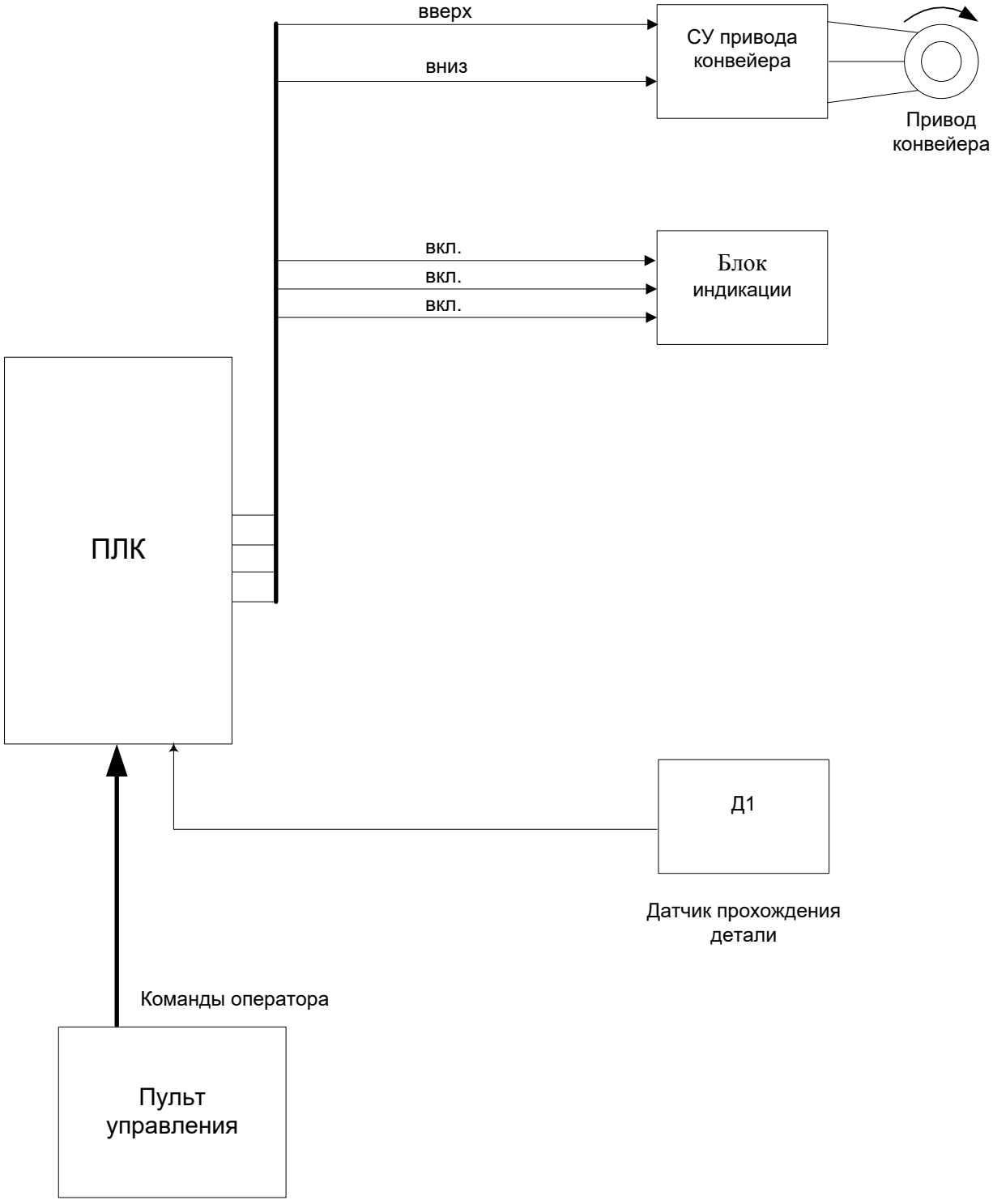



Рис. 3.8. Структурная схема СУ к упражнению С-4

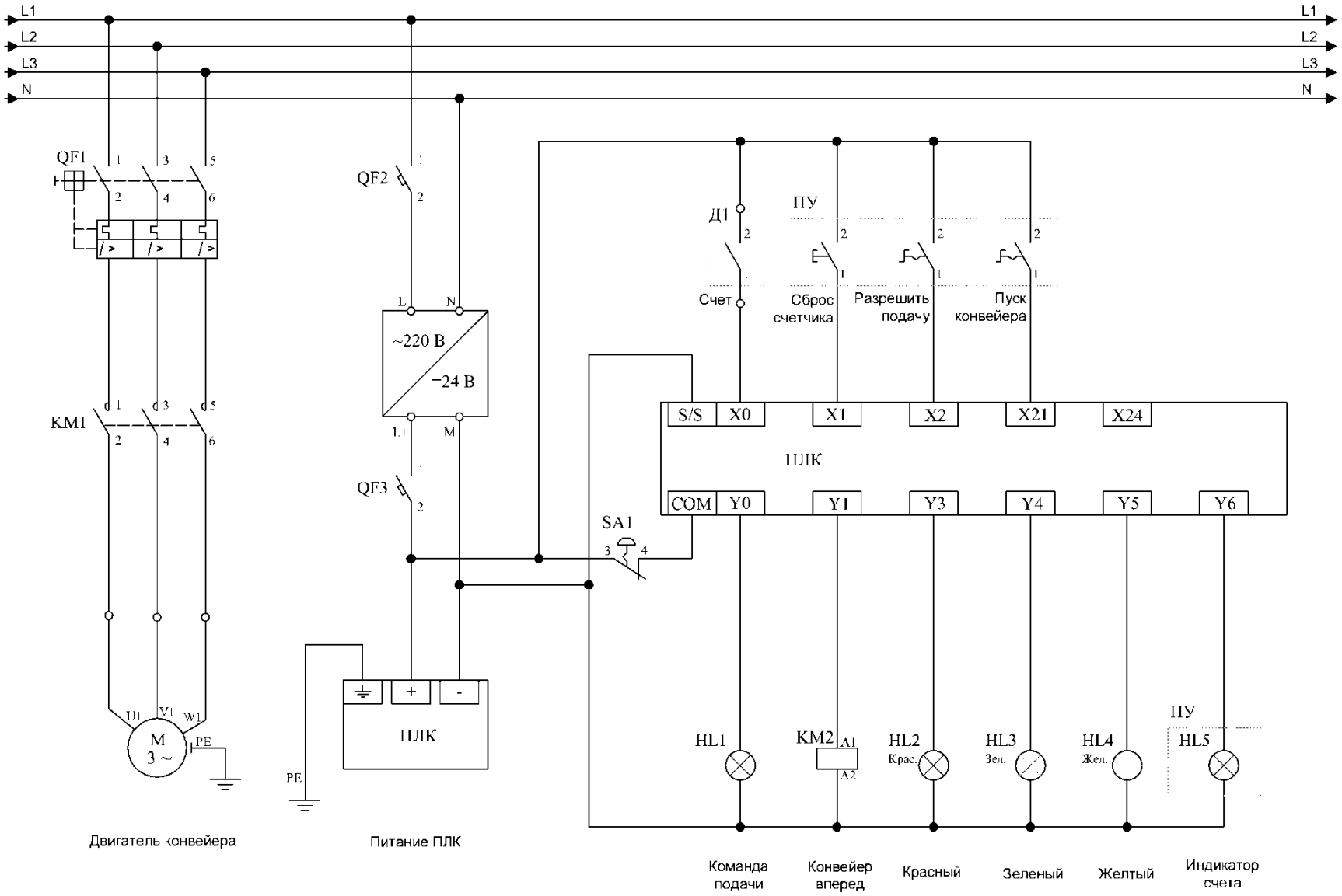



Рис. 3.9. Схема подключения СУ (упражнение С-4)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2, QF3 – автоматические выключатели
KM1 – контактор для управления двигателем
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
Д1 – датчик обнаружения детали
HL1, HL2, HL3, HL4, HL5 – световая сигнализация

**Контрольные вопросы**

1. Какие виды таймеров Вы знаете, и как они работают?
2. Синтаксис инициализации таймера.
3. Что такое дискретность внутреннего времени таймера?
4. Перечислите виды погрешностей таймеров.
5. Назначение и виды счетчиков.
6. Синтаксис инициализации счетчика.
7. Какие бывают счетчики по способу обработки импульсов?
8. Усовершенствуйте записанную программу из упражнения С-1 таким образом, чтобы в процессе открытия двери горела лампа *Зеленого цвета* (Y6), а в процессе закрытия двери горела лампа *Красного цвета* (Y5).
9. Измените записанную программу из упражнения С-3 так, чтобы сигналы *Зеленого (Y1)* и *Красного (Y0) цвета* мерцали попеременно с интервалом 1с.

# ЛАБОРАТОРНАЯ РАБОТА №4

## Управление сигналами светофоров и устройством звуковой сигнализации при обнаружении пешехода или автомобиля. Управление технологическим оборудованием в соответствии с сигналами датчиков

**Цель работы**

Самостоятельно создать программное обеспечение для заданных в упражнениях D-2 и D-6 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

1. **Упражнение D-2 – Управление сигналами светофоров и устройством звуковой сигнализации при обнаружении пешехода или автомобиля.**

На рисунке 4.1 показано виртуальное оборудование: 2 светофора (для пешехода и автомобиля) и устройство звуковой сигнализации, которое срабатывает в случае, если автомобиль более 10с находится в так называемой активной зоне ворот (участок между датчиками *X2* и *X3*). Ворота на рисунке 4.1 не показаны, т.к. задачей упражнения является управление светофорами и устройством звуковой сигнализации, но не открытием-закрытием ворот. Датчики *Человек у ворот* (*X0)* и *Человек за воротами (X1), Автомобиль у ворот (X2)* и *Автомобиль за воротами (X3)* призваны контролировать местонахождение пешехода и автомобиля соответственно.

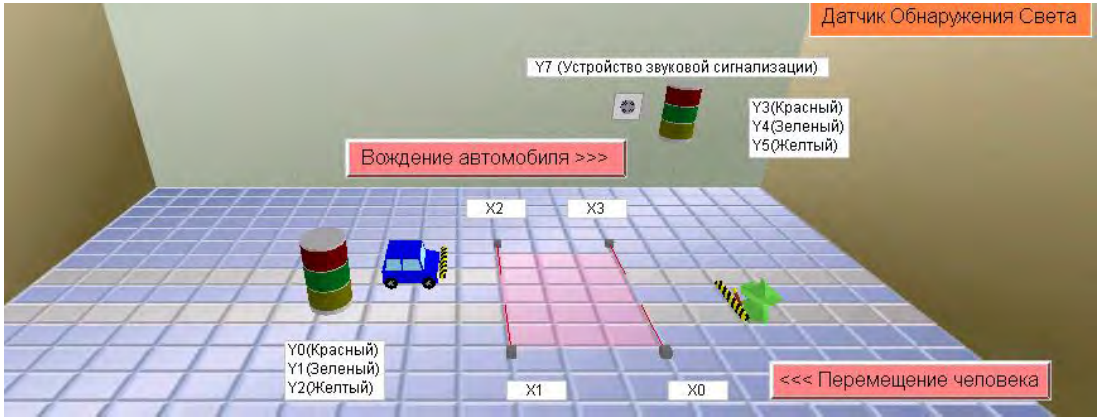


Рис. 4.1. Виртуальное оборудование к упражнению D-2

Для управления предложенным в упражнении D-2 виртуальным оборудованием предлагается система управления, структурная схема которой показана на рисунке 4.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление виртуальным оборудованием. Блок датчиков регистрирует местонахождение пешехода или автомобиля, выходные сигналы датчиков поступают на входы контроллера, программно обрабатываются, и с выходов ПЛК осуществляется управление сигналами светофоров и устройством сигнализации.

Схема подключения СУ приведена на рисунке 4.3.

**Задание**

Теоретические сведения о таймерах, использование которых необходимо при написании программы управления предложенным оборудованием, приведены в разделе «Язык релейно-контактных схем (LD)» главе «Программирование таймера. Команда TIMER» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

1.1. Разработать управляющую программу для управления сигналами светофоров и устройством звуковой сигнализации при обнаружении пешехода или автомобиля с учетом следующих условий:

Со стороны человека

a) Когда датчик *Человек у ворот (X0)* обнаруживает человека, загорается сигнал *Зеленого цвета (Y1)* светофора.
b) Спустя 5 секунд после того, как датчик *Человек за воротами (X1)* обнаружит прохождение человека, сигнал *Зеленого цвета (Y1)* гаснет.

Со стороны автомобиля

a) Когда датчик *Автомобиль у ворот (X2)* обнаруживает автомобиль, загорается сигнал *Зеленого цвета (Y4)* светофора.
b) Спустя 5 секунд после того, как датчик *Автомобиль за воротами (X3)* обнаружит прохождение автомобиля, проблесковый сигнал *Зеленого цвета (Y4)* гаснет.
c) Если автомобиль не пересечет пространство между датчиками *X2* и *X3* в течение 10 секунд, то загорается сигнал *Красного цвета (Y3)* светофора и срабатывает *Устройство звуковой сигнализации (Y7)*.

d) Как только автомобиль минует датчик *Автомобиль за воротами (X3)*, сигнал *Красного цвета* светофора *(Y3)* гаснет и *Устройство звуковой сигнализации (Y7)* отключается.

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.**

1.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению D-2

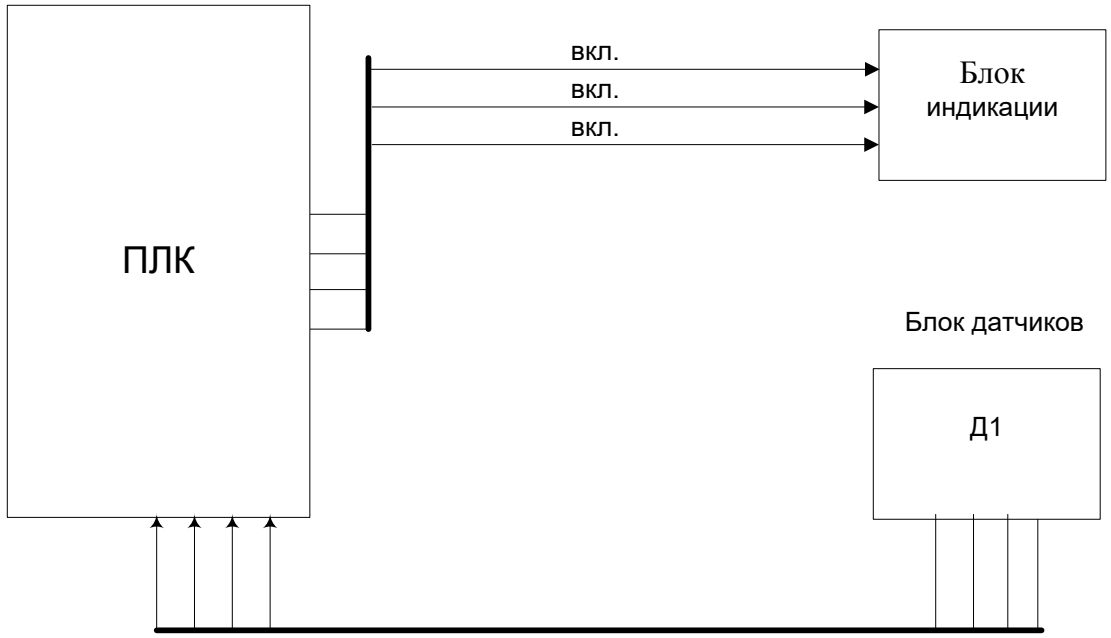



Рис. 4.2. Структурная схема СУ к упражнению D-2

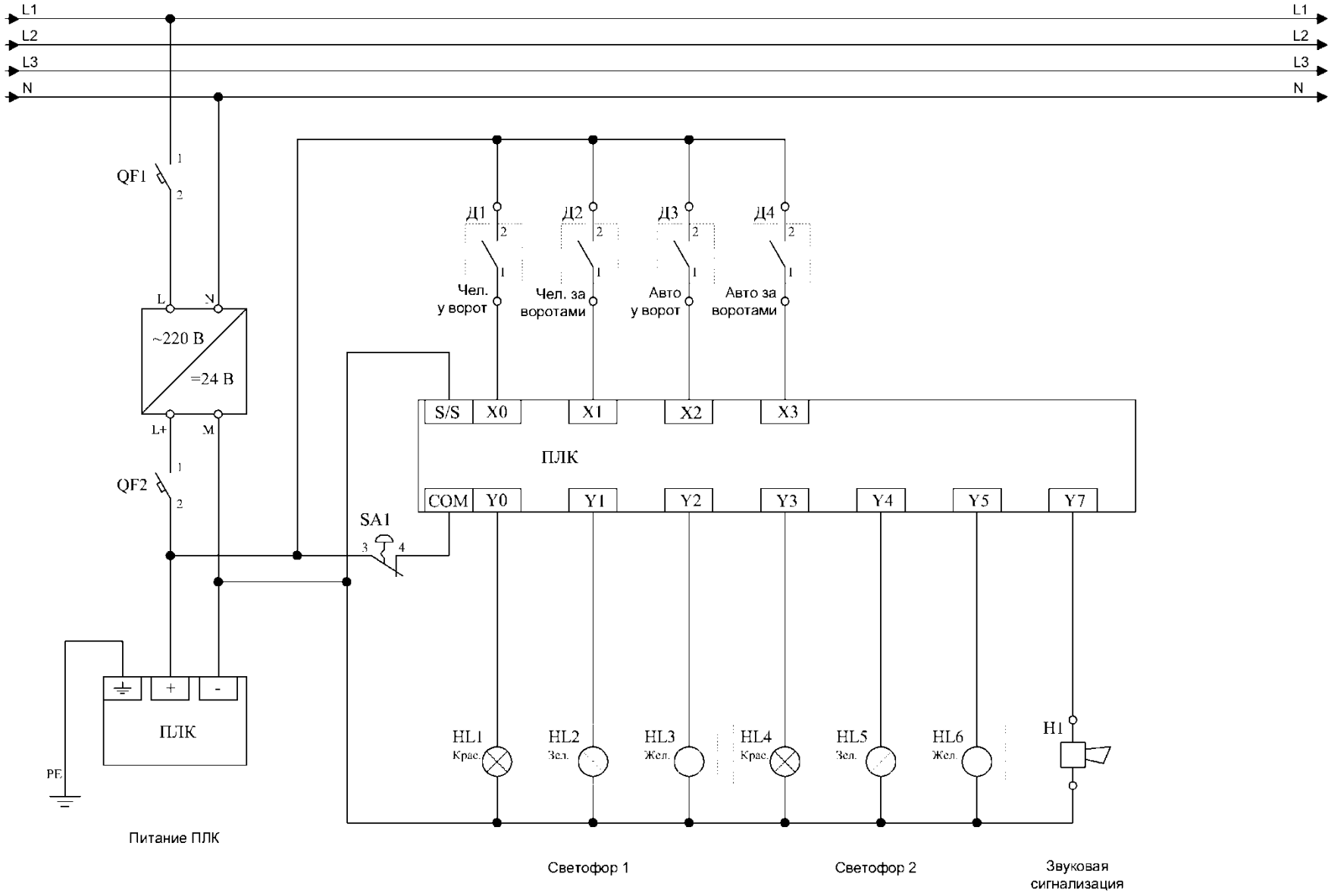



Рис. 4.3. Схема подключения СУ (упражнение D-2)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматические выключатели
Д1, Д2 – датчики обнаружения человека
Д3, Д4 – датчики обнаружения автомобиля
H1 – звонок (звуковая сигнализация)
HL1…HL6 – световая сигнализация
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

## 2. Упражнение D-6 – Управление технологическим оборудованием в соответствии с сигналами датчиков.

На рисунке 4.4 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** задает для робота с выхода *Y7* управляющий сигнал подачи детали на конвейер при условии, что робот находится в отправной точке (нормально открытый контакт входа *X5* контроллера замкнут). Датчик *X0*, обнаружив деталь, выдает сигнал *Верхний конвейер вперед (Y0)*. Датчик *X1,* обнаружив деталь, выдает сигнал *Средний конвейер вперед (Y2)* и останавливает верхний конвейер с выхода *Y0* контроллера. Датчик *X2*, обнаружив деталь, выдает сигнал *Нижний конвейер вперед (Y4)* и останавливает средний конвейер с выхода *Y2* контроллера. Когда датчик *X3* обнаруживает деталь, он выдает сигнал на останов нижнего конвейера с выхода *Y4* контроллера и *Команду подачи (Y7)* для робота, при условии, что последний находится в отправной точке.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 4.5.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

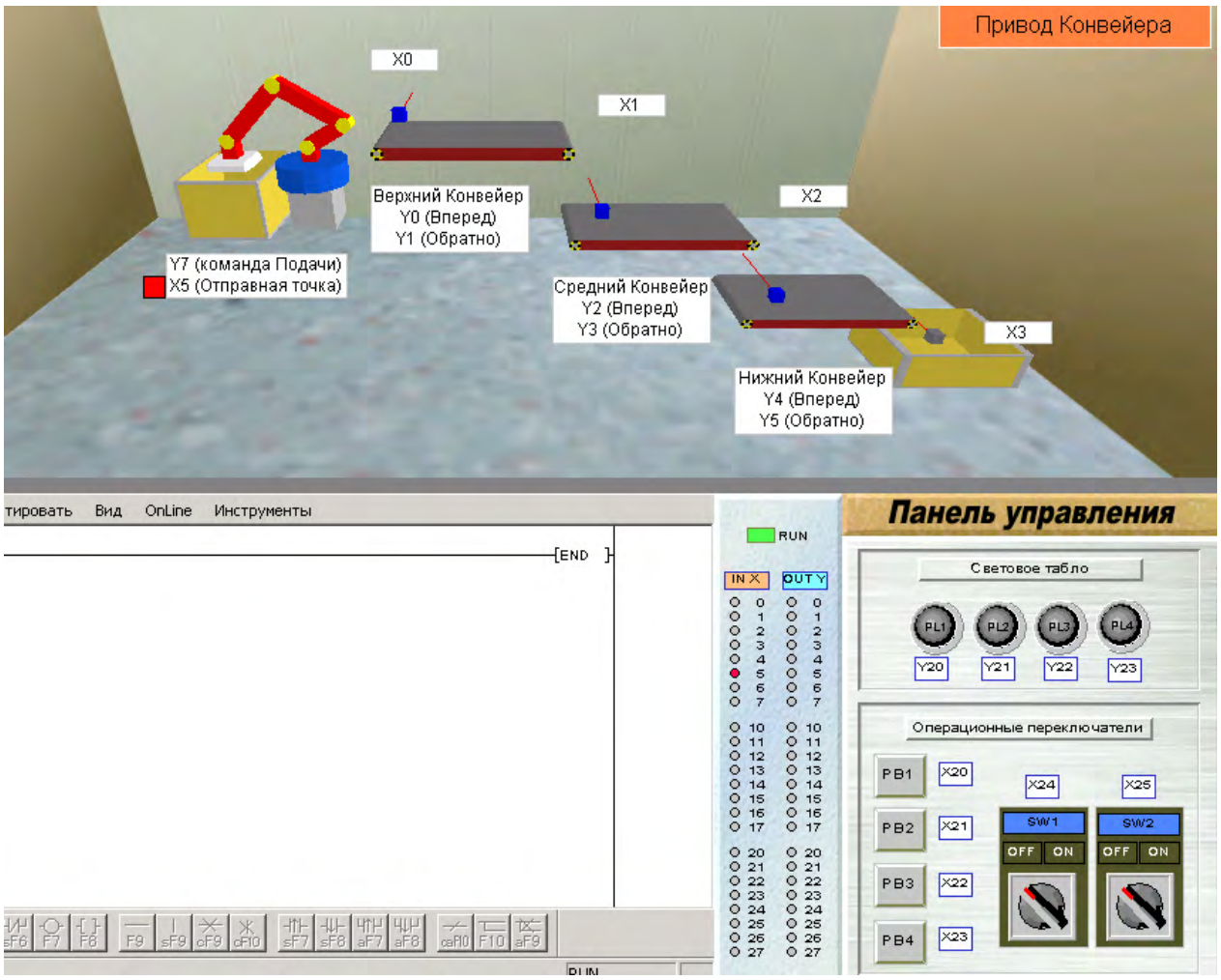



Рис. 4.4. Панель управления и виртуальное оборудование к упражнению D-6

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунке 4.6, 4.7.

**Задание**

2.1. Разработать управляющую программу для реализации поставленной задачи управления технологическим оборудованием в соответствии с сигналами датчиков с учетом следующих условий:

a) Когда на **Панели управления** нажата *кнопка PB1 (X20)*, роботу выдается *Команда Подачи (Y7)*, при условии, что робот находится в *Отправной точке (X5)*. При отпускании кнопки *PB1 (X20) Команда Подачи (Y7)* защелкивается на период, пока робот не возвратится в *Отправную точку (X5)*.

b) Когда датчик *X0* обнаруживает деталь, выдается команда *Верхний конвейер вперед (Y0)*.

c) Когда датчик *X1* обнаруживает деталь, выдается команда *Средний конвейер вперед (Y2)* и осуществляется останов верхнего конвейера *(Y0)*.

d) Когда датчик *X2* обнаруживает деталь, подается команда *Нижний конвейер вперед (Y4)* и осуществляется останов среднего конвейера *(Y2)*.

e) Когда датчик *X3* обнаруживает деталь, подается команда на останов нижнего конвейера *(Y4)*.
f) Если датчик *X3* перешел в состояние *ON*, то роботу выдается *Команда подачи (Y7)* и выполняется передача очередной детали на конвейер, при условии, что робот находится в *Отправной точке (X5)*.

2.2. Выполнить проверку соответствия программы заданным условиям посредством 3D -**графической имитации**.

2.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению D-6

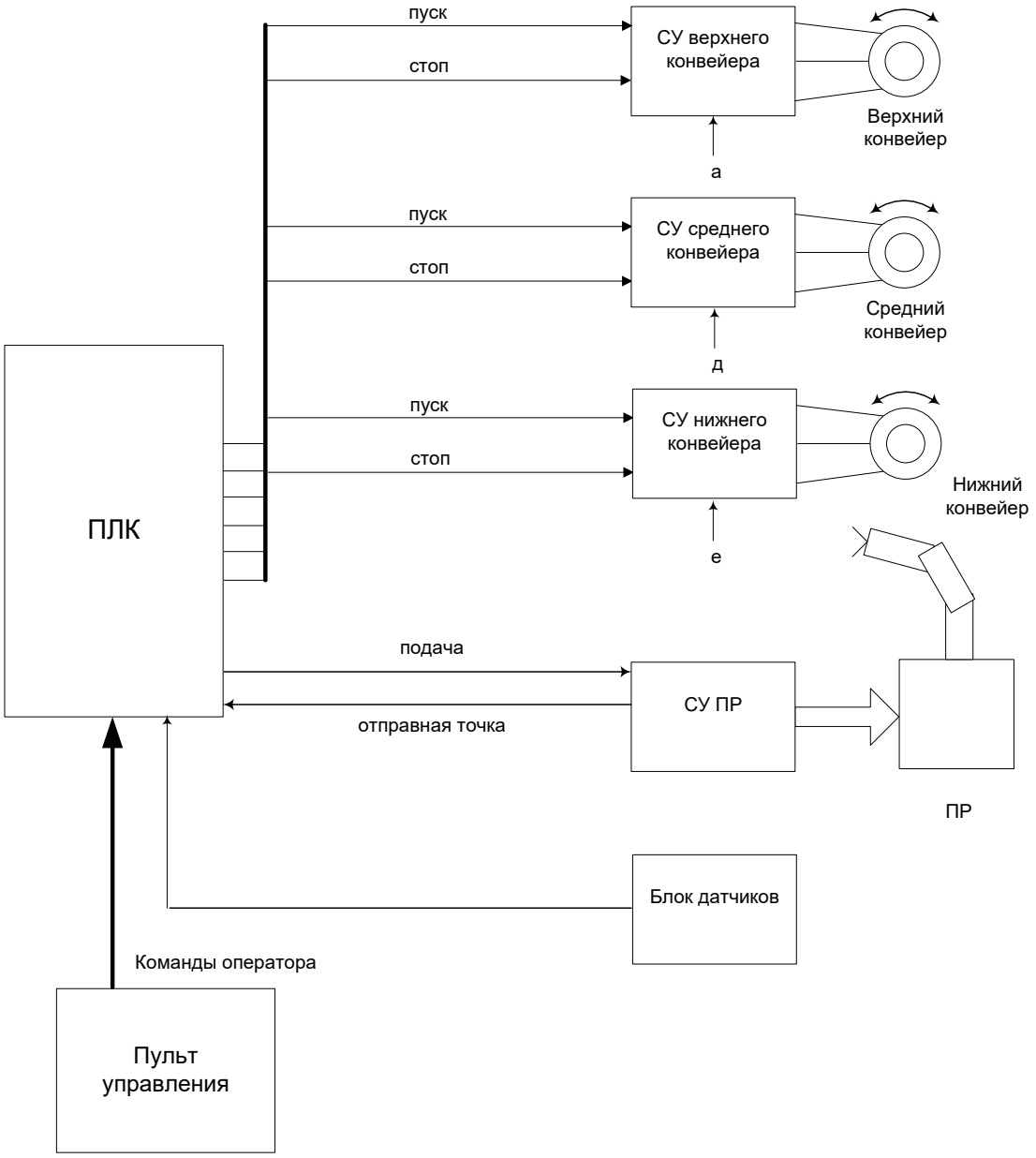



Рис. 4.5. Структурная схема СУ к упражнению D-6

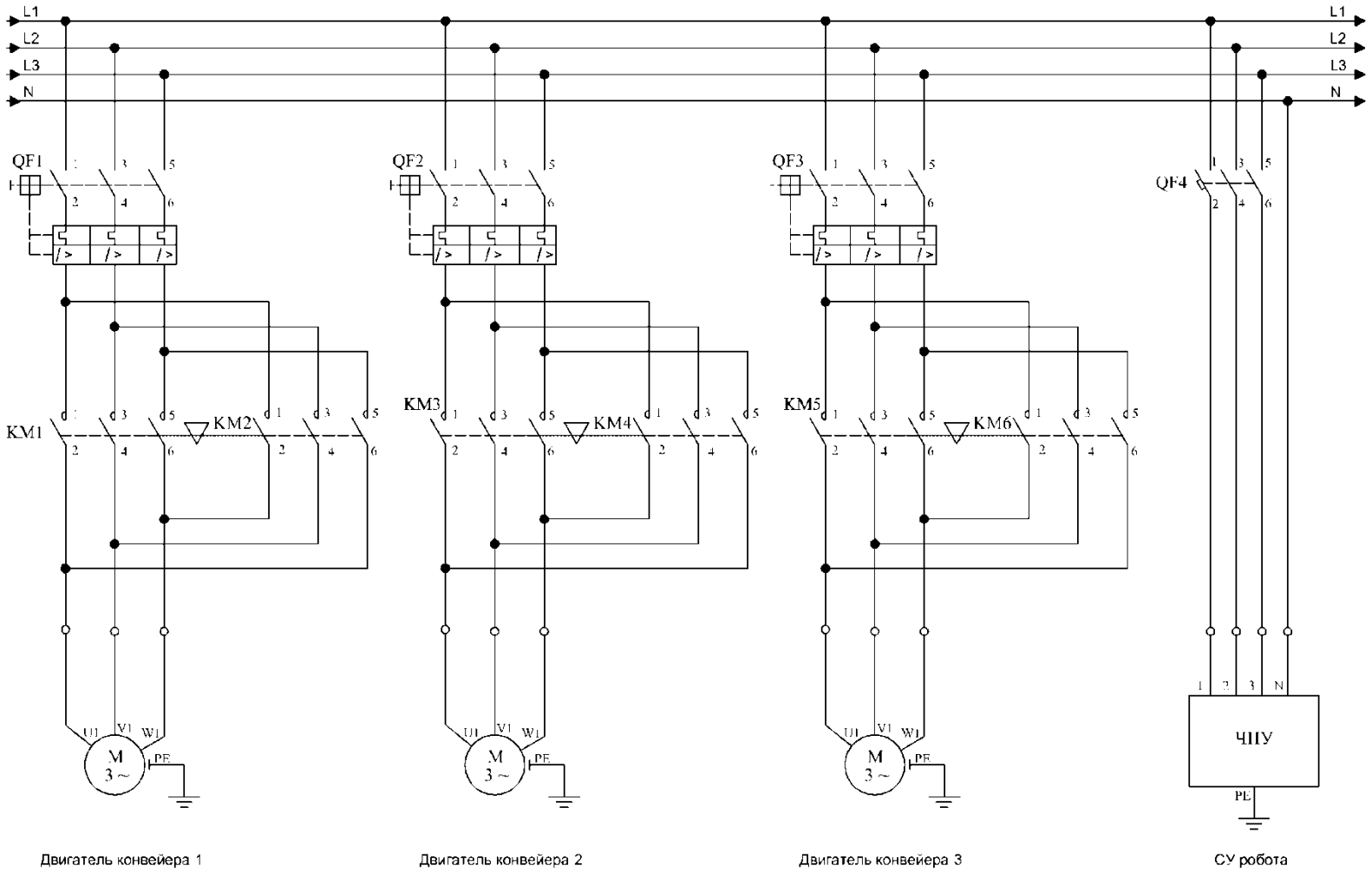



Рис. 4.6. Схема подключения оборудования (упражнение D-6)

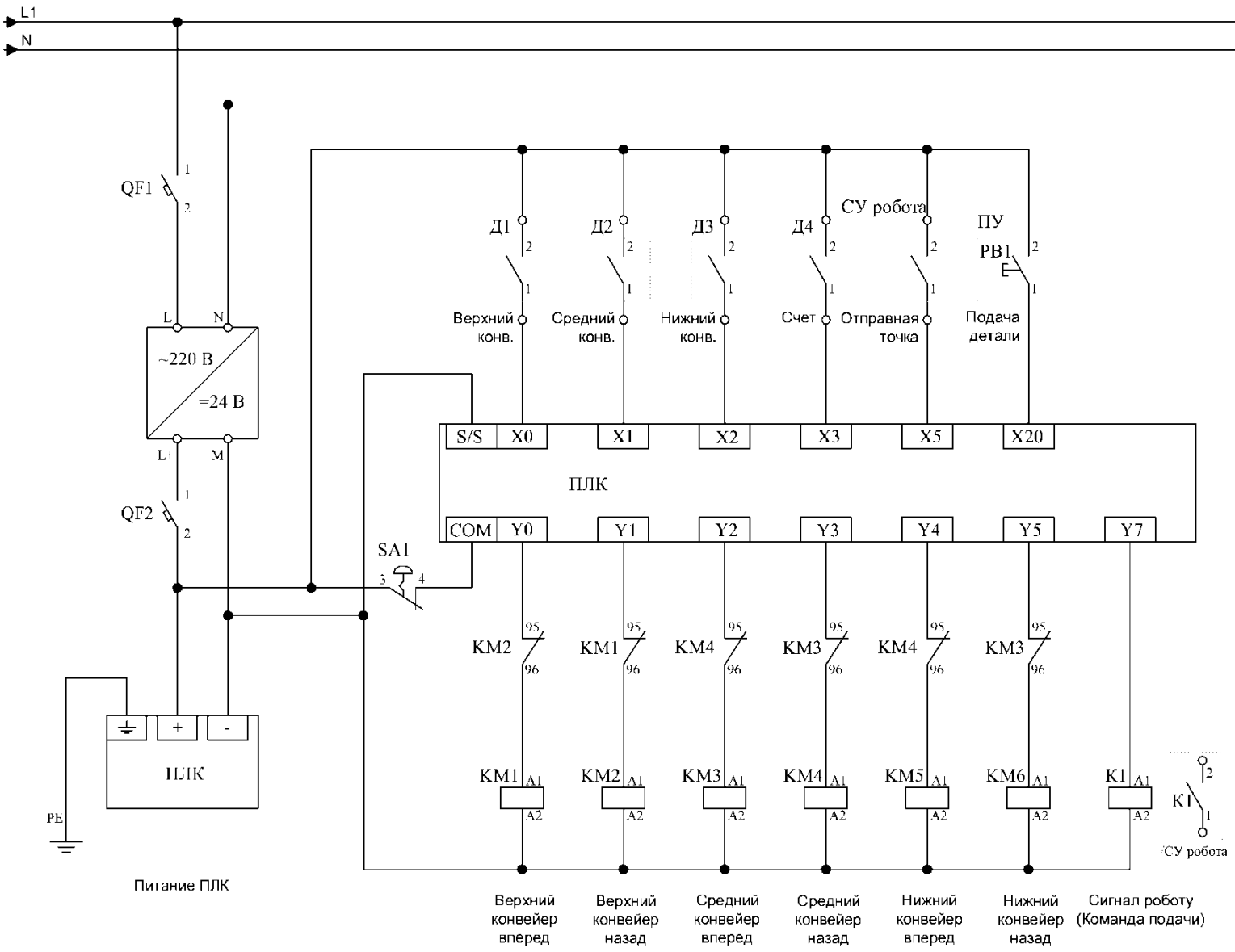



Рис. 4.7. Схема подключения входов-выходов ПЛК (упражнение D-6)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2, QF3 – автоматические выключатели для двигателей с тепловыми реле
QF4 – автоматический выключатель
KM1 – контактор для управления двигателем верхнего конвейера (вперед)
KM2 – контактор для управления двигателем верхнего конвейера (реверс)
KM3 – контактор для управления двигателем среднего конвейера (вперед)
KM4 – контактор для управления двигателем среднего конвейера (реверс)
KM5 – контактор для управления двигателем нижнего конвейера (вперед)
KM6 – контактор для управления двигателем нижнего конвейера (реверс)
K1 – реле для подачи управляющего сигнала роботу
Д1…Д4 – датчики положения детали
PB1 – кнопка на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. В чем заключается различие между нормально закрытым и нормально открытым контактами?
2. Какой из контактов, нормально закрытый или нормально открытый, должен соответствовать кнопке аварийного останова системы и почему?
3. Опишите структуру ПЛК и назначение его компонентов, как существующих физически, так и моделируемых.
4. С помощью каких устройств при необходимости может быть расширен ПЛК?
5. Усовершенствуйте созданную в упражнении D-2 программу таким образом, чтобы устройство звуковой сигнализации срабатывало, если пешеход находится в активной зоне ворот (участок между датчиками *X0* и *X1*) более 20с.
6. Измените созданную в упражнении D-6 программу таким образом, чтобы технологическое оборудование прекращало работу в случае, когда поддон с деталями будет наполнен (вместимость поддона 7 деталей).

# ЛАБОРАТОРНАЯ РАБОТА №5

## Управление сигналами светофора. Сортировка деталей по размеру

### Цель работы

Самостоятельно создать программное обеспечение для заданных в упражнениях D-3 и D-4 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

### 1. Упражнение D-3 – Управление сигналами светофора.

На рисунке 5.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК и кнопкой *PB1,* по нажатию которой стартует процесс управления сигналами светофора.

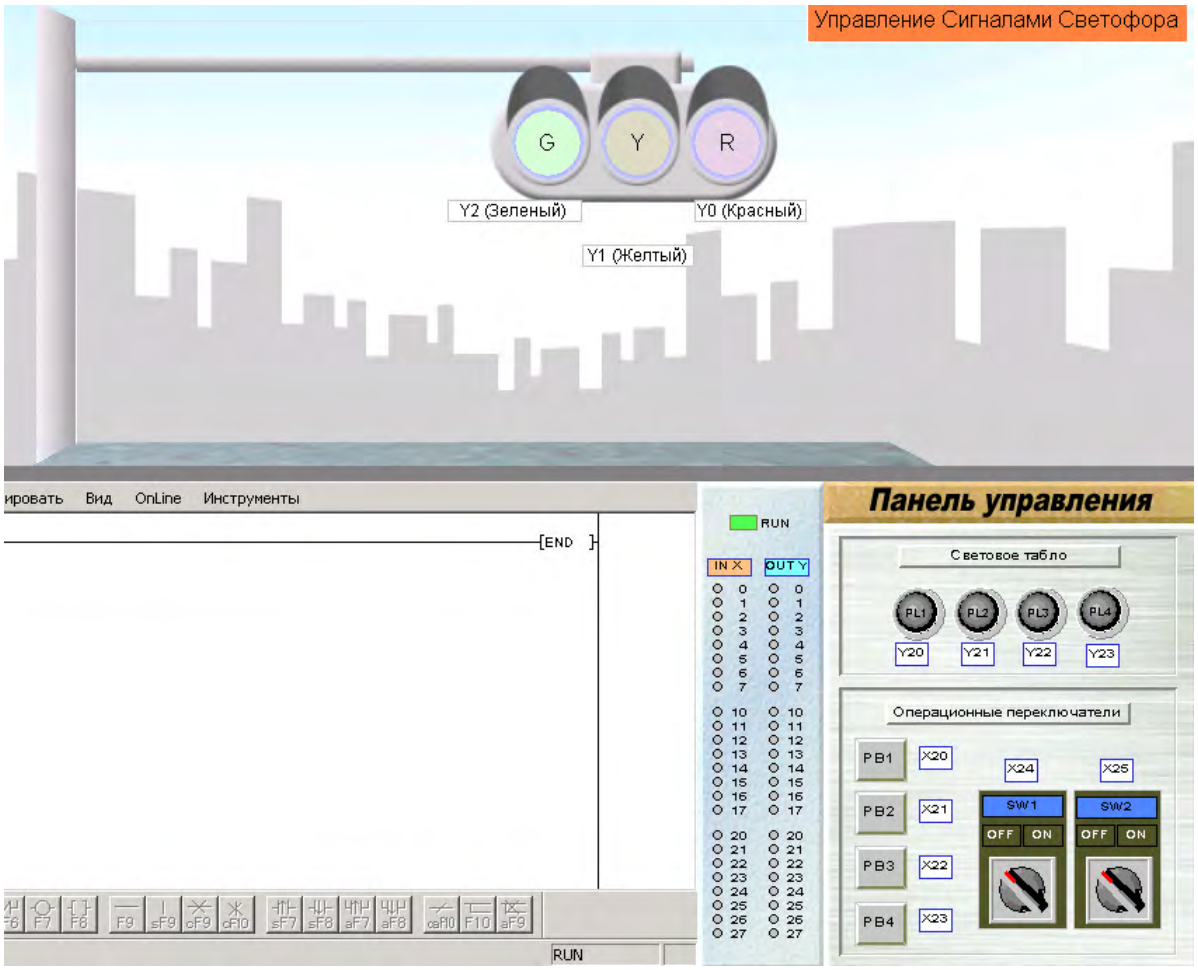



Рис. 5.1. Панель управления и виртуальное оборудование к упражнению D-3

Структурная схема СУ для реализации поставленной задачи управления сигналами светофора показана на рисунке 5.2.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление сигналами светофора.

Схема подключения СУ приведена на рисунке 5.3.

**Задание**

Теоретические сведения о таймерах, использование которых необходимо при написании программы управления предложенным оборудованием, приведены в разделе «Язык релейно-контактных схем (LD)» главе «Программирование таймера. Команда TIMER» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

1.1. Разработать управляющую программу для реализации поставленной задачи управления сигналами светофора с учетом следующих условий:
   a) Процесс стартует по нажатию на **Панели управления** кнопки *PB1 (X20)*.
   b) После запуска в течение 10 секунд горит сигнал *Красного цвета (Y0)*.
   c) Сигнал *Красного цвета (Y0)* гаснет через 10 секунд. Загорается сигнал *Желтого цвета (Y1)* и горит в течение 5 секунд.
   d) Сигнал *Желтого цвета (Y1)* гаснет, загорается сигнал *Зеленого цвета (Y2)* и горит в течение 10 секунд, затем гаснет.
   e) Процедура функционирования продолжается, начиная с шага b).

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**.

1.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению D-3

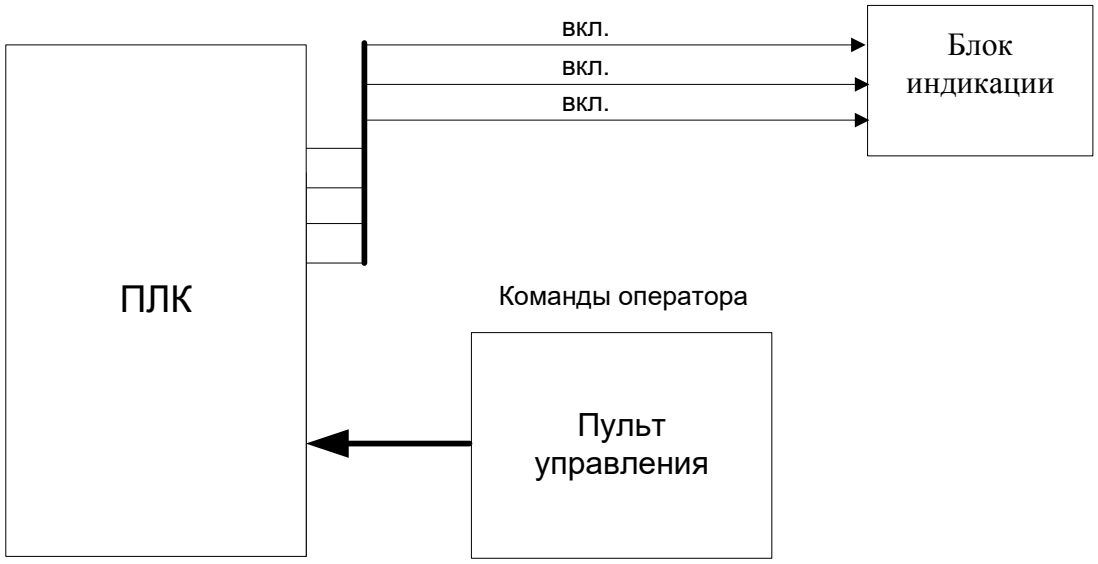



Рис. 5.2. Структурная схема СУ к упражнению D-3

**Схема подключения СУ** (упражнение D-3)

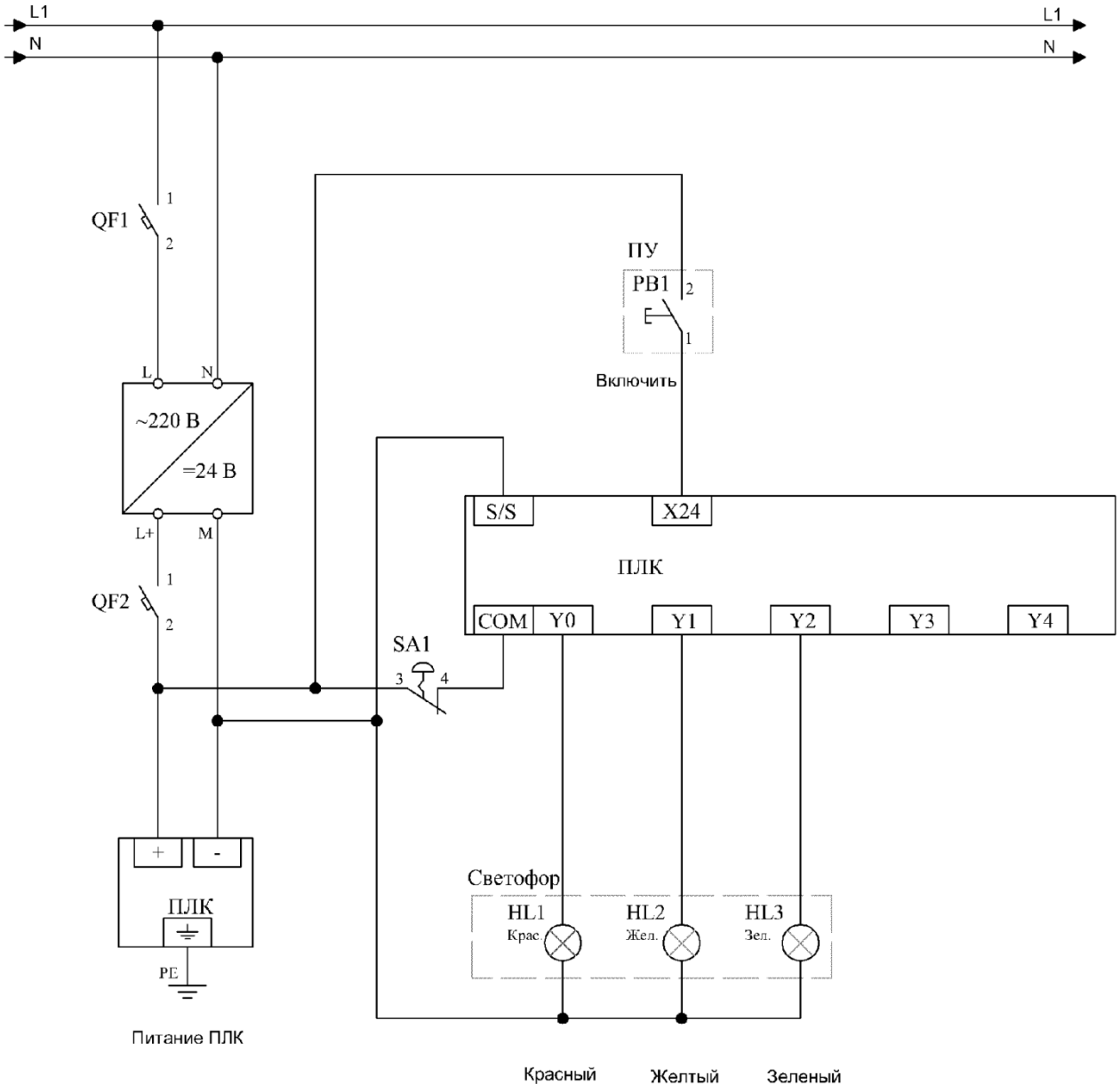



Рис. 5.3. Схема подключения СУ (упражнение D-3)

L1 – однофазный источник питания (220В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматические выключатели
HL1 – лампа красного цвета
HL2 – лампа желтого цвета
HL3 – лампа зеленого цвета
PB1 – кнопка включения на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

## 2. Упражнение D-4 – Сортировка деталей по размеру.

На рисунке 5.4 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

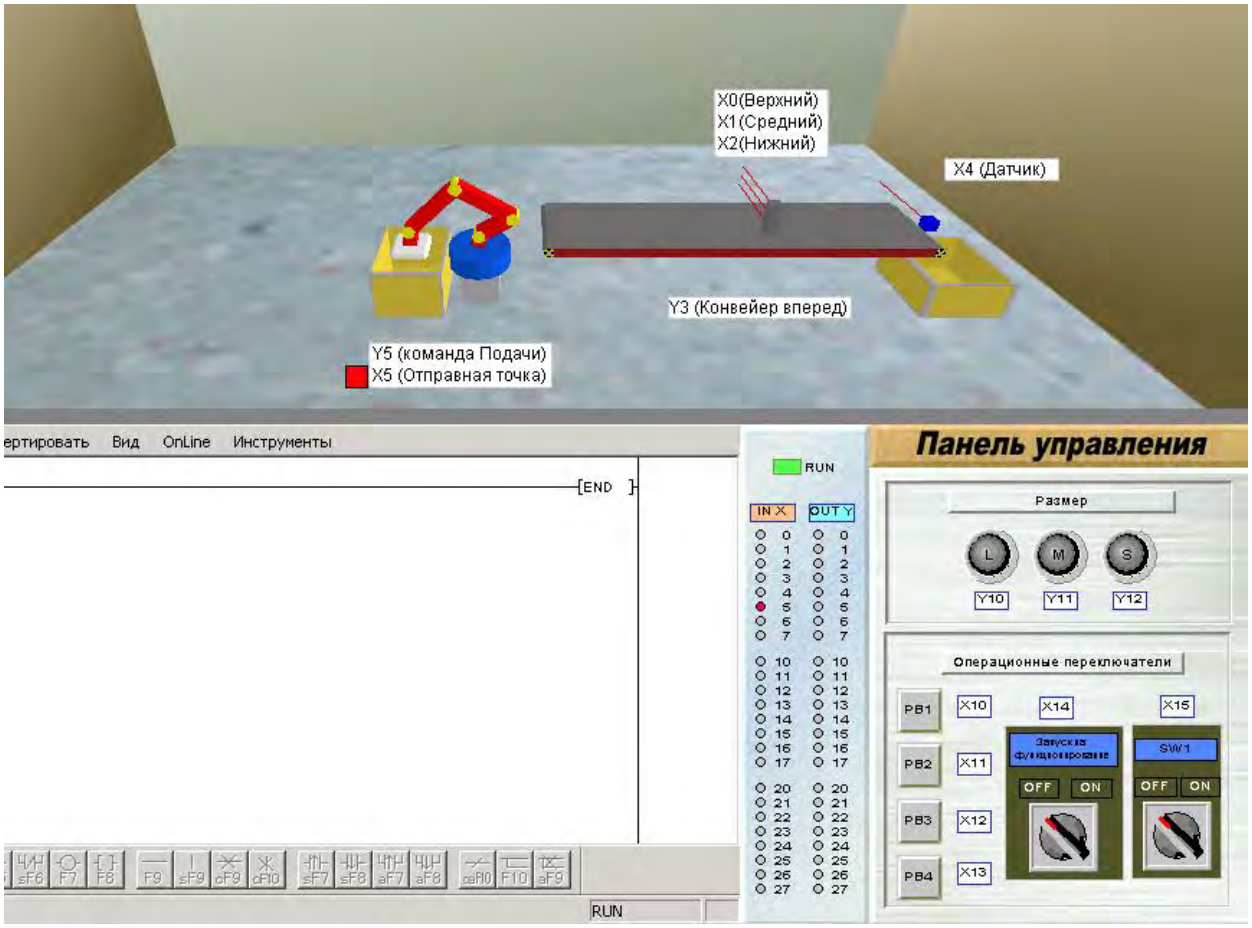



Рис. 5.4. Панель управления и виртуальное оборудование к упражнению D-4

Кнопка *PB1* (вход *X10* контроллера) на **Панели управления** в состоянии *ON* задает для робота с выхода *Y5* управляющий сигнал подачи детали на конвейер. Тумблер *X14* управляет пуском-остановом конвейера с выхода *Y3* контроллера в положениях *ON/OFF* соответственно. Датчики *X0, X1, X2* фиксируют прохождение деталей большой, средней и малой величины соответственно. По сигналу от соответствующего датчика загорается одна из ламп индикации на **Панели управления:** *Y10* для больших, *Y11* для средних и *Y12* для малых деталей. Лампы гаснут, когда датчик *X4* зафиксирует прохождение детали по конвейеру.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 5.5.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера.

Обработка сигналов датчиков осуществляется программно. Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 5.6, 5.7.

**Задание**

Теоретические сведения по операторам управления по фронтам входных сигналов LDP/LDF приведены в разделе «Язык релейно-контактных схем (LD)» главе «Основные команды» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами», по операторам генерации одиночных импульсов по фронтам входных сигналов PLF/PLS – в главе «Программирование одиночных импульсов. Команды (PLF) и (PLS)», по программированию внутреннего реле – в главе «Программирование внутреннего реле».

2.1. Разработать управляющую программу для реализации поставленной задачи сортировки деталей в соответствии с сигналами различных датчиков с учетом следующих условий:
   a) Когда на **Панели управления** нажата кнопка *PB1 (X10)*, переходит в *ON Команда подачи (Y5)* для робота. Если кнопка *PB1 (X10)* отпущена, *Команда подачи (Y5)* переходит в *OFF*.
   b) Когда на **Панели управления** тумблер *Запуск на функционирование (X14)* установлен в положение *ON*, подается команда *Конвейер вперед (Y3)*. Когда тумблер *Запуск на функционирование (X14)* установлен в положение *OFF*, подается команда на останов конвейера *(Y3)*.
   c) Большие, средние или мелкие детали, переносимые конвейером, сортируются в соответствии с состоянием входов датчиков *Верхний (X0), Средний (X1)* и *Нижний (X2)* и затем загораются соответствующие лампы *Большой (Y10), Средней (Y11)* или *Малой величины (Y12)*.
   d) Лампы загораются сразу же после того, как датчик классифицирует деталь и гаснут, когда деталь проходит датчик *(X4)*.

2.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.**

2.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению D-4

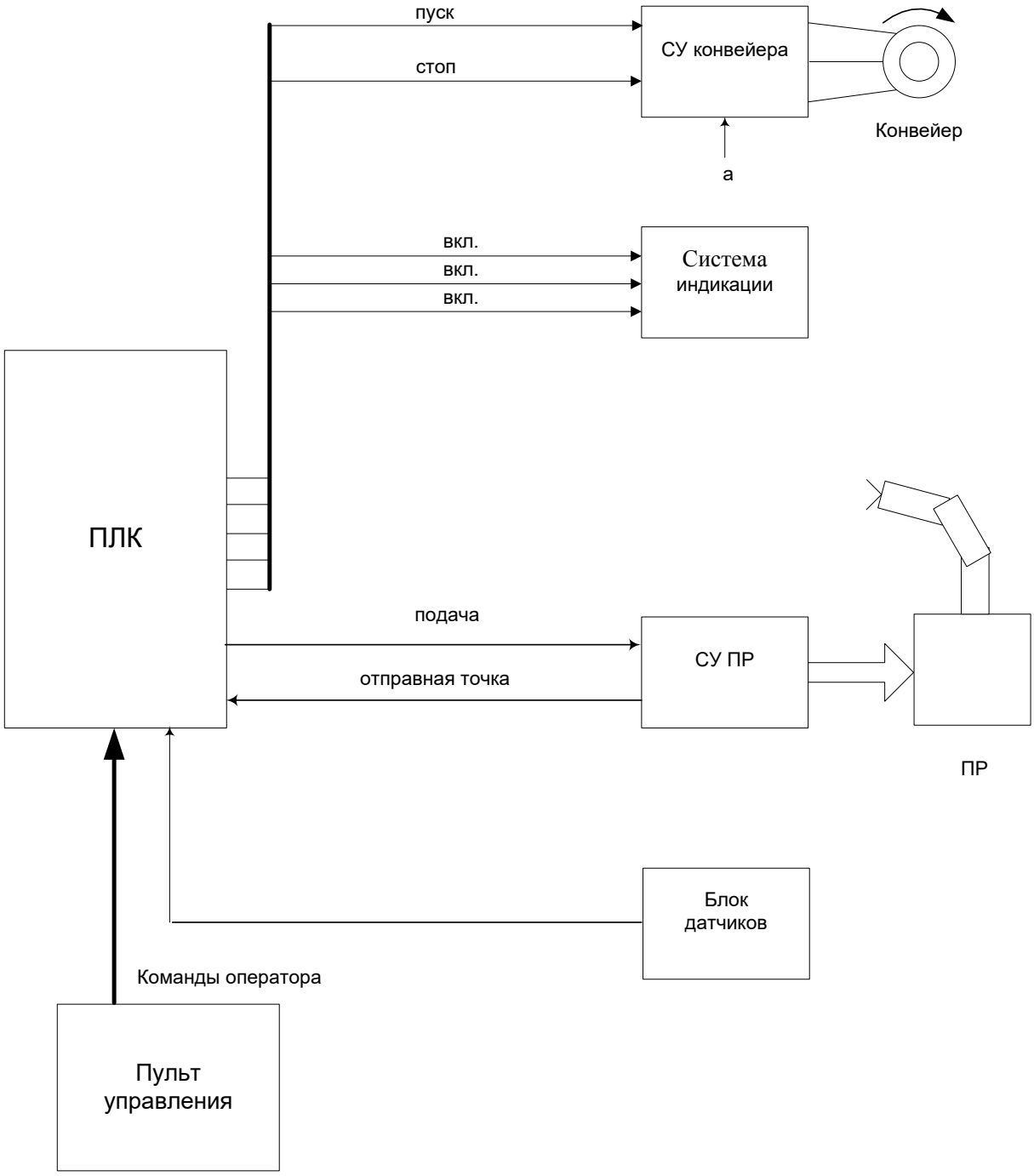



Рис. 5.5. Структурная схема СУ к упражнению D-4

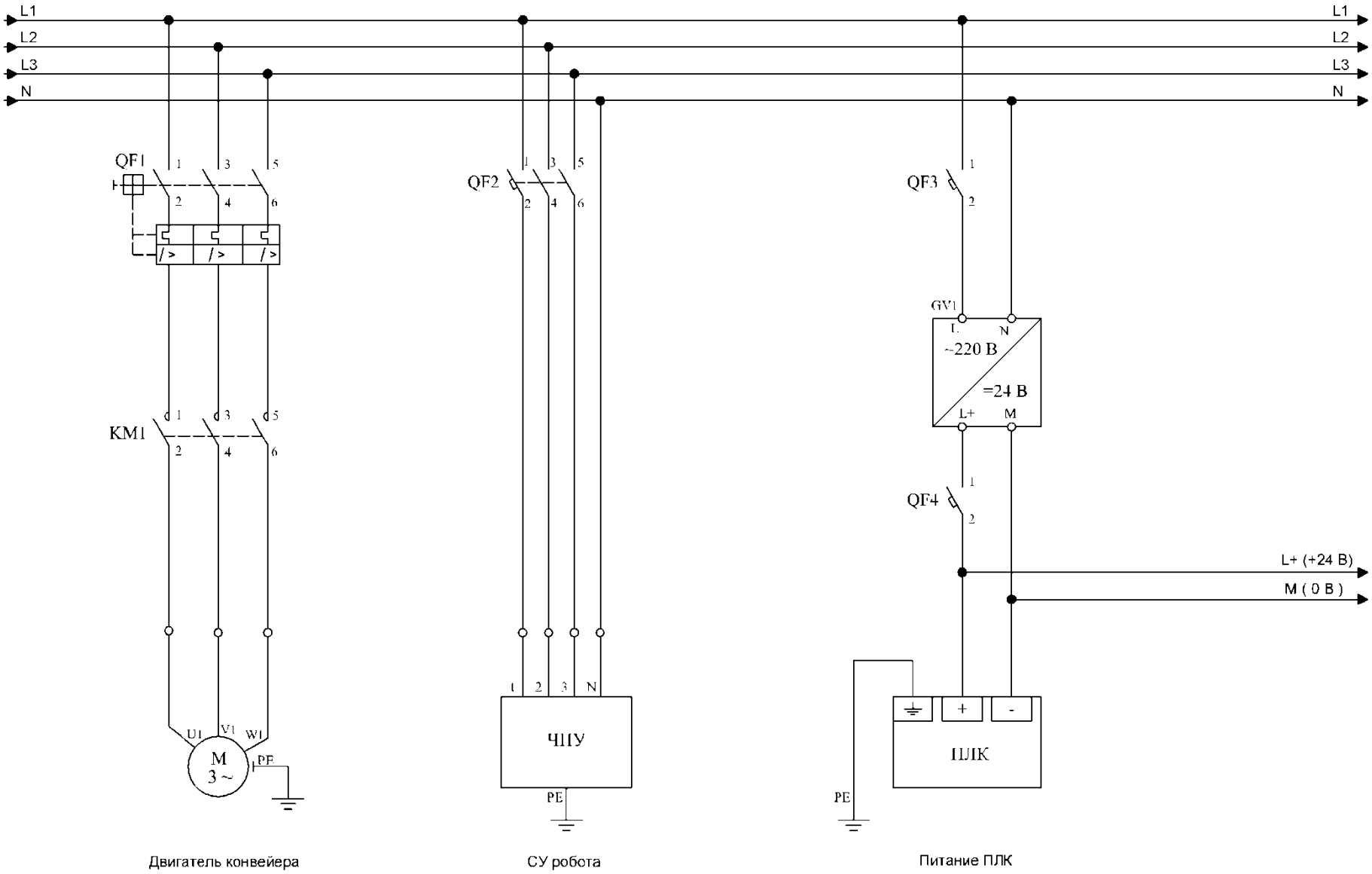



Рис. 5.6. Схема подключения оборудования (упражнение D-4)

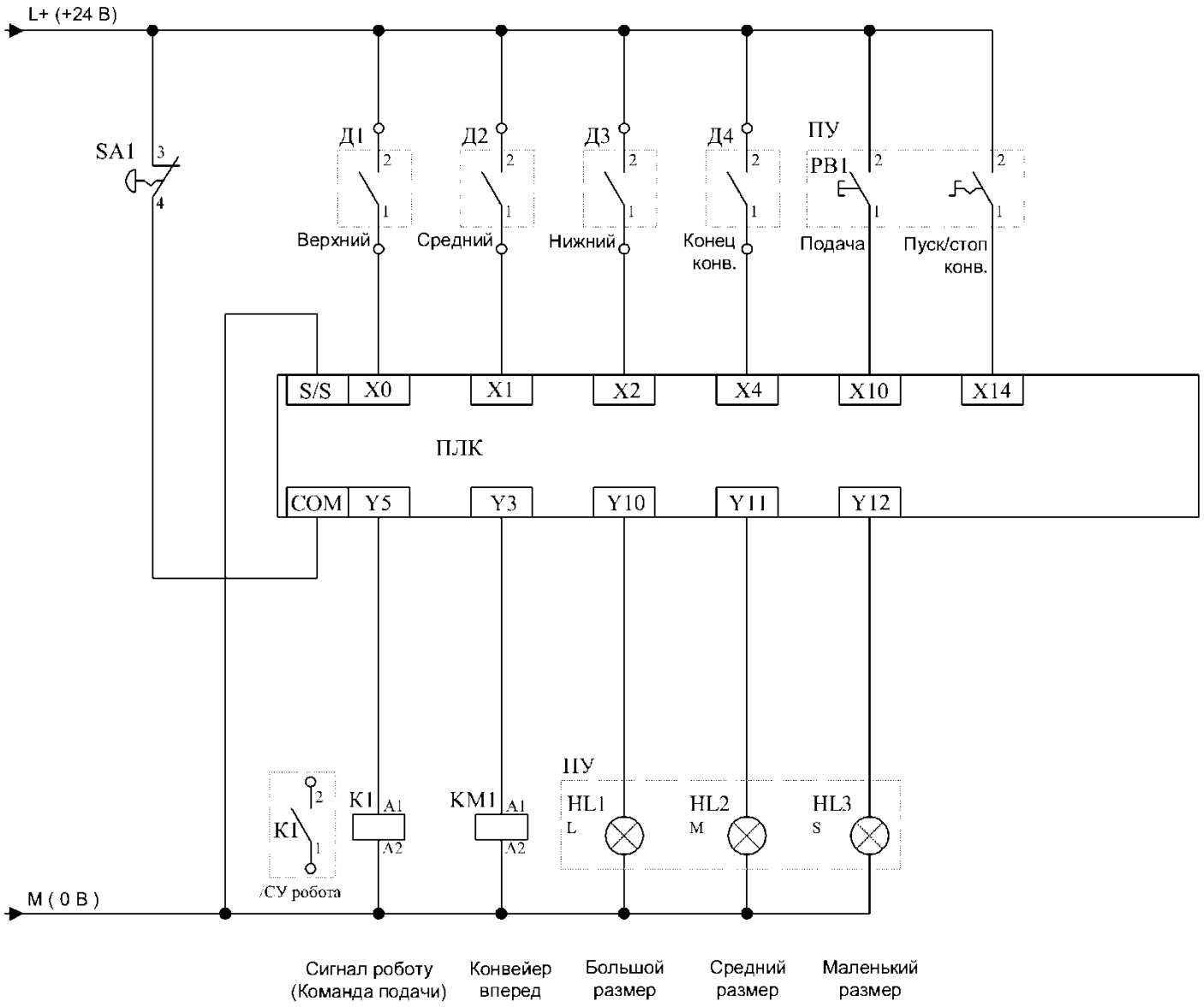



Рис. 5.6. Схема подключения ПЛК (упражнение D-4)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF4 – автоматические выключатели
KM1 – контактор для управления двигателем
K1 – реле для подачи управляющего сигнала роботу
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
Д1…Д4 – датчики положения детали
PB1 – кнопка на панели управления
«Запуск на функционирование»– тумблер на панели управления

**Контрольные вопросы**

1. Опишите 4 основных шага в цикле обработки программы контроллером.
2. Почему очень важное значение имеет обнуление данных при инициализации системы?
3. Из каких параметров складывается понятие «быстродействие ПЛК»?
4. Какие варианты решения проблемы недостаточно высокого быстродействия Вы знаете?
5. Каково оптимальное время продолжительности импульса и почему?
6. Измените созданную в упражнении D-3 программу таким образом, чтобы после подачи сигнала на запуск функционирования (нажатие кнопки *PB1 (X20)* на **Панели управления**) цикл управления сигналами светофора отрабатывался 3 раза, а затем все сигналы гасли.
7. Усовершенствуйте созданную в упражнении D-4 программу таким образом, чтобы управляющий сигнал подачи детали на конвейер (*Y5*) для робота срабатывал только при условии, что робот находится в *Отправной точке (X5)*.

# ЛАБОРАТОРНАЯ РАБОТА №6

## Переключение сигналов светофора в ответ на нажатие кнопки. Сортировка деталей по размеру

**Цель работы**

Самостоятельно создать программное обеспечение для заданных в упражнениях Е-1 и Е-2 СУ, используя изученные ранее основные инструкции, применяемые при программировании контроллеров.

1. **Упражнение Е-1 – Переключение сигналов светофора в ответ на нажатие кнопки.**

На рисунке 6.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК и кнопкой *X10,* по нажатию которой загорается индикатор *Y10* и стартует процесс управления сигналами светофора.

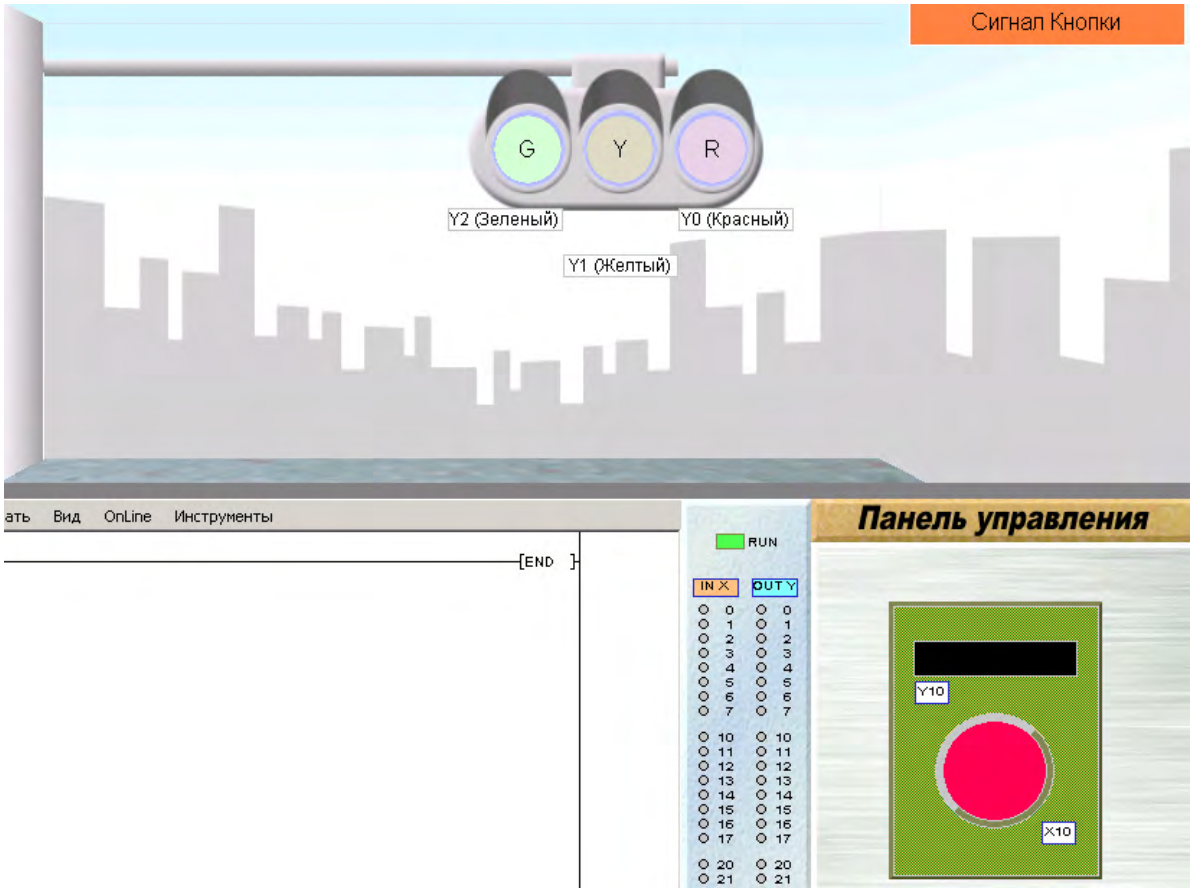



Рис. 6.1. Панель управления и оборудование к упражнению Е-1

Структурная схема СУ для реализации поставленной задачи управления сигналами светофора показана на рисунке 6.2.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление сигналами светофора.

Схема подключения СУ приведена на рисунке 6.3.

**Задание**

Теоретические сведения по назначению таймеров, их классификации, инициализации и примерам использования приведены в разделе «Язык релейно-контактных схем (LD)» главе «Программирование таймера. Команда TIMER» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами».

1.1. Разработать управляющую программу для реализации поставленной задачи переключения сигналов светофора в ответ на нажатие кнопки с учетом следующих условий:

a) Сигнал *Красного цвета (Y0)* мерцает с интервалом в одну секунду.
b) Если на **Панели управления** будет нажата кнопка *(X10)*, загорается индикатор «*Пожалуйста, обождите» (Y10),* который продолжает гореть в течение 5с после того, как кнопка *(X10)* будет отпущена.
c) После того, как через 5с индикатор «*Пожалуйста, обождите» (Y10)* погаснет, запускается функционирование сигналов в соответствие с последовательностью приведенной в пунктах с d) по g).
d) Пока в течение 5 секунд горит индикатор *(Y10)*, продолжает мерцать сигнал *Красного цвета (Y0)*.
e) Сигнал *Красного цвета (Y0)* отключается, и в течение 5 секунд горит сигнал *Желтого цвета (Y1)*.
f) После погасания сигнала *Желтого цвета (Y1)* в течение 10 секунд горит сигнал *Зеленого цвета (Y2)*.
g) После погасания сигнала *Зеленого цвета (Y2)* включается сигнал *Красного цвета (Y0)* и мерцает с интервалом в одну секунду. Повторяется функционирование, начиная с пункта a).

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы произвести по следующим пунктам:

a) Функционирование в исходном состоянии
Результат ► сигнал *Красного цвета (Y0)* мерцает с интервалом в одну секунду (1 секунда в *ON* и 1 секунда в *OFF*).
b) Подтверждение работы кнопки *(X10)*
Результат ►когда кнопка *(X10)* нажата, в течение 5 секунд горит индикатор с надписью «*Пожалуйста, обождите» (Y10)* и сигнал *Красного цвета (Y0)* продолжает мерцать.
c) Функционирование после погасания надписи «*Пожалуйста, обождите» (Y10)*
Результат ► в течение 5 секунд горит сигнал *Желтого цвета (Y1)*.
d) Функционирование после погасания сигнала *Желтого цвета (Y1)*
Результат ► в течение 10 секунд горит сигнал *Зеленого цвета (Y2)*.
e) Функционирование после погасания сигнала *Зеленого цвета (Y2)*

Результат ►сигнал *Красного цвета (Y0)* мерцает с интервалом в одну секунду.

Для того чтобы иметь возможность повторить функционирование, следует инициализировать экран посредством щелчка на дистанционном управлении по кнопке *Reset*.

1.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению Е-1

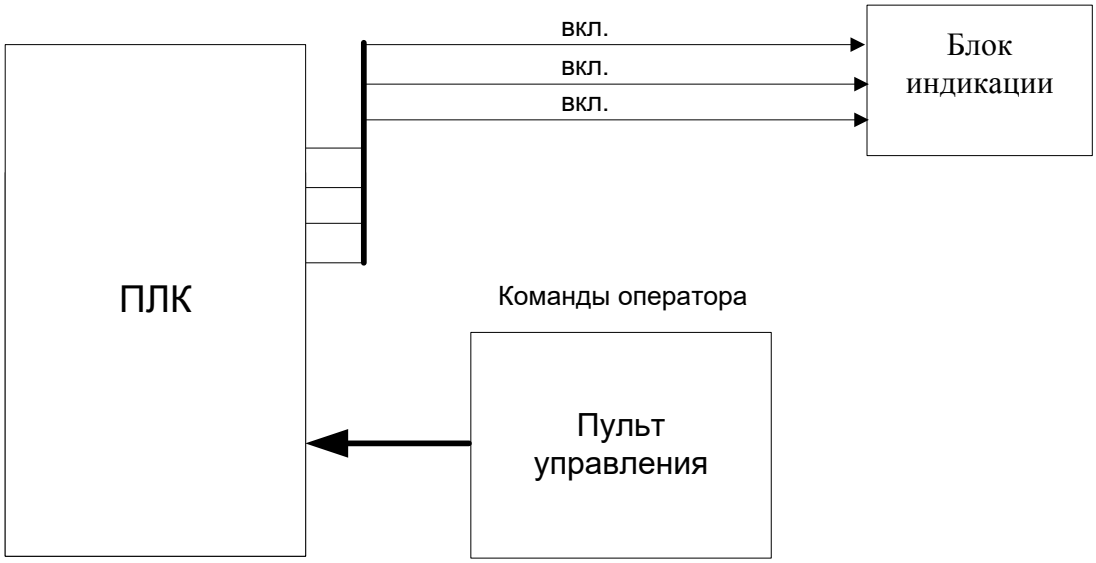



Рис. 6.2. Структурная схема СУ к упражнению Е-1

**Схема подключения СУ** (упражнение Е-1)

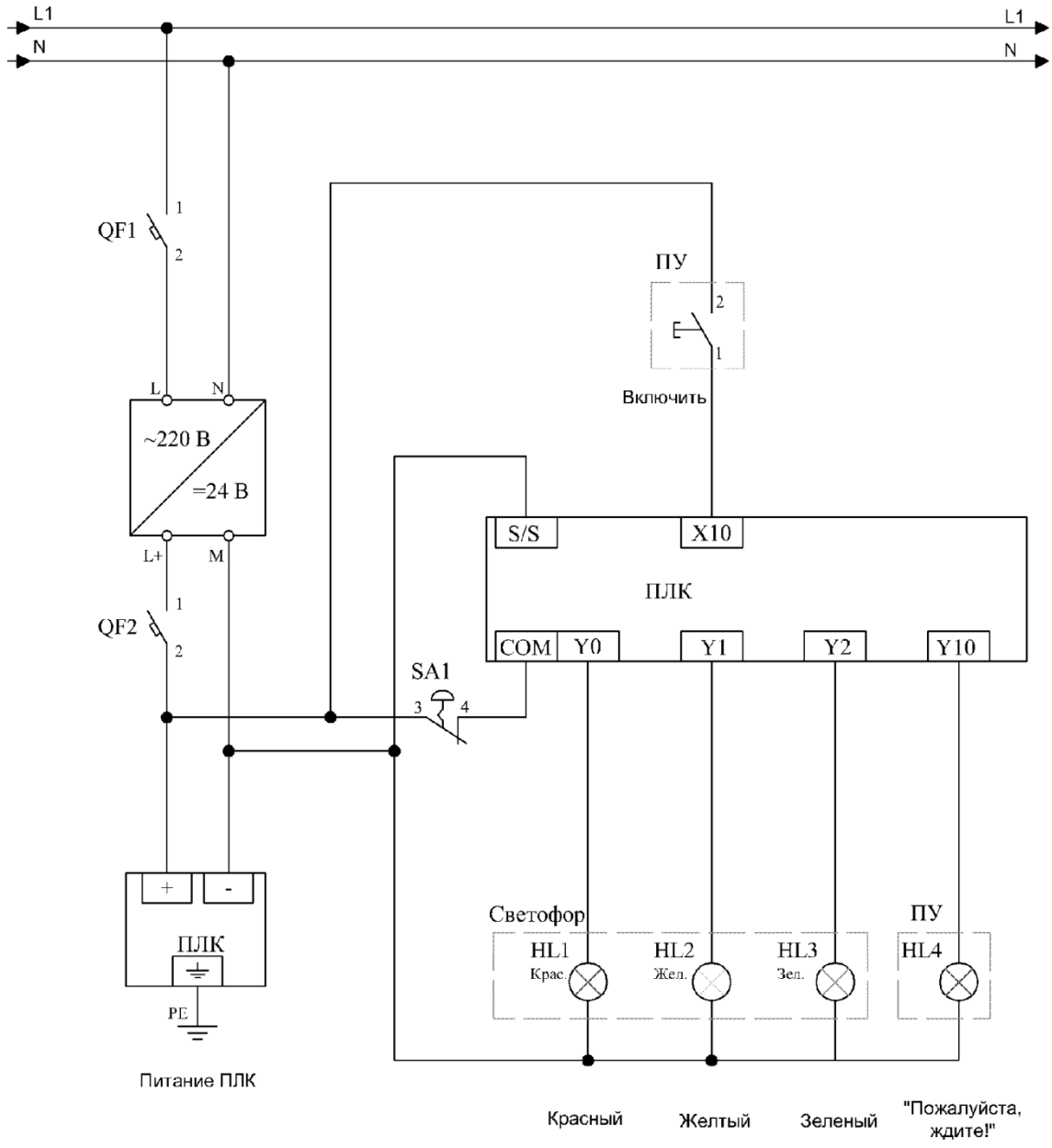



Рис. 6.3. Схема подключения СУ (упражнение Е-1)

L1 – однофазный источник питания (220В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматические выключатели
HL1 – лампа красного цвета
HL2 – лампа желтого цвета
HL3 – лампа зеленого цвета
HL4 – индикация на панели управления
X10 – кнопка включения на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

## 2. Упражнение E-2 – Сортировка деталей по размеру.

На рисунке 6.4 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

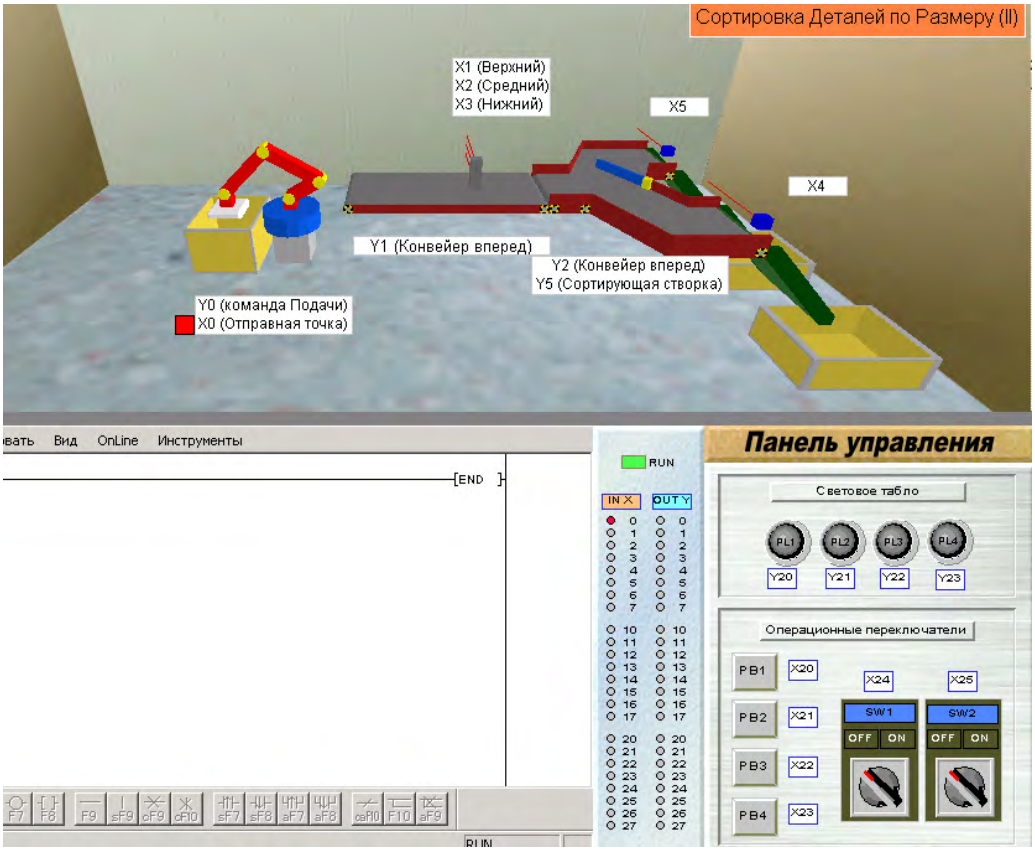



Рис. 6.4. Панель управления и виртуальное оборудование к упражнению E-2

Тумблер *X24* на **Панели управления** управляет пуском-остановом конвейеров с выходов *Y1 и Y2* контроллера в положениях *ON/OFF* соответственно. Кнопка *PB1* (вход *X20* контроллера) задает для робота с выхода *Y0* управляющий сигнал подачи детали на конвейер при условии, что робот находится в отправной точке *X0*. Сортировка деталей по размерам осуществляется по сигналам датчиков *Верхнего* (*X1)* и *Нижнего (X3),* которые фиксируют прохождение деталей большой и малой величины соответственно.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 6.5.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров. Запуск программы промышленного робота осуществляется с выхода

контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 6.6, 6.7.

**Задание**

2.1. Разработать управляющую программу для реализации поставленной задачи распределения деталей в определенное место в соответствии с их размерами с учетом следующих условий:

a) Когда на **Панели управления** тумблер *SW1 (X24)* переведен в *ON*, конвейеры *Y1* и *Y2* движутся. Когда тумблер *SW1 (X24)* установлен в *OFF*, конвейеры останавливаются.

b) Когда на **Панели управления** нажата кнопка *PB1 (X20)*, роботу выдается *команда Подачи (Y0)* при условии, что робот находится в отправной точке *X0.*.

c) Робот подает детали больших и малых размеров.

d) Детали больших размеров направляются к тыловому конвейеру, а мелкие детали направляются к переднему конвейеру. Размер деталей определяется входом датчиков *Верхнего (X1)* и *Нижнего (X3)*, установленных на конвейере. Направление движения деталей по конвейеру задается положением *Сортирующей створки (Y5)*.

**ПРЕДОСТЕРЕЖЕНИЕ**

Для корректного выполнения программы необходимо задать условие, что только одна деталь может одновременно находиться на конвейере.

2.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы произвести по следующим пунктам:

a) На **Панели управления** установите в *ON* тумблер *SW1 (X24)*
Результат ► конвейеры перемещаются вправо.

b) На **Панели управления** нажмите кнопку *PBI (X20)*
Результат ►робот выполняет подачу детали.

c) Сортировка деталей
Результат ► большая деталь переносится к тыловому конвейеру. Маленькая деталь переносится к переднему конвейеру.

d) Повторение операций
Результат ►когда нажимается кнопка *PB1 (X20),* осуществляется повтор функционирования, начиная с пункта b).

2.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению Е-2

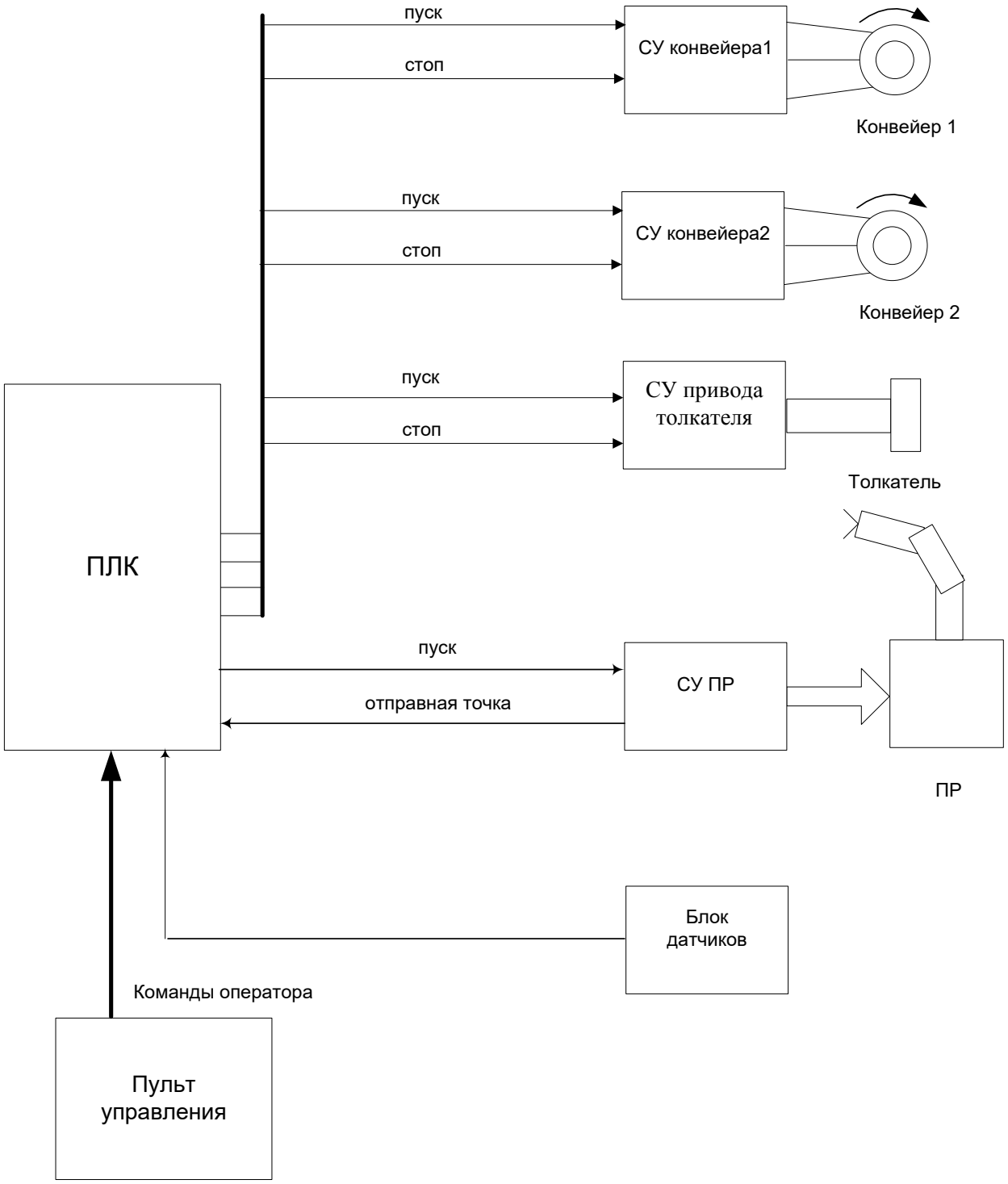



Рис. 6.5. Структурная схема СУ к упражнению Е-2

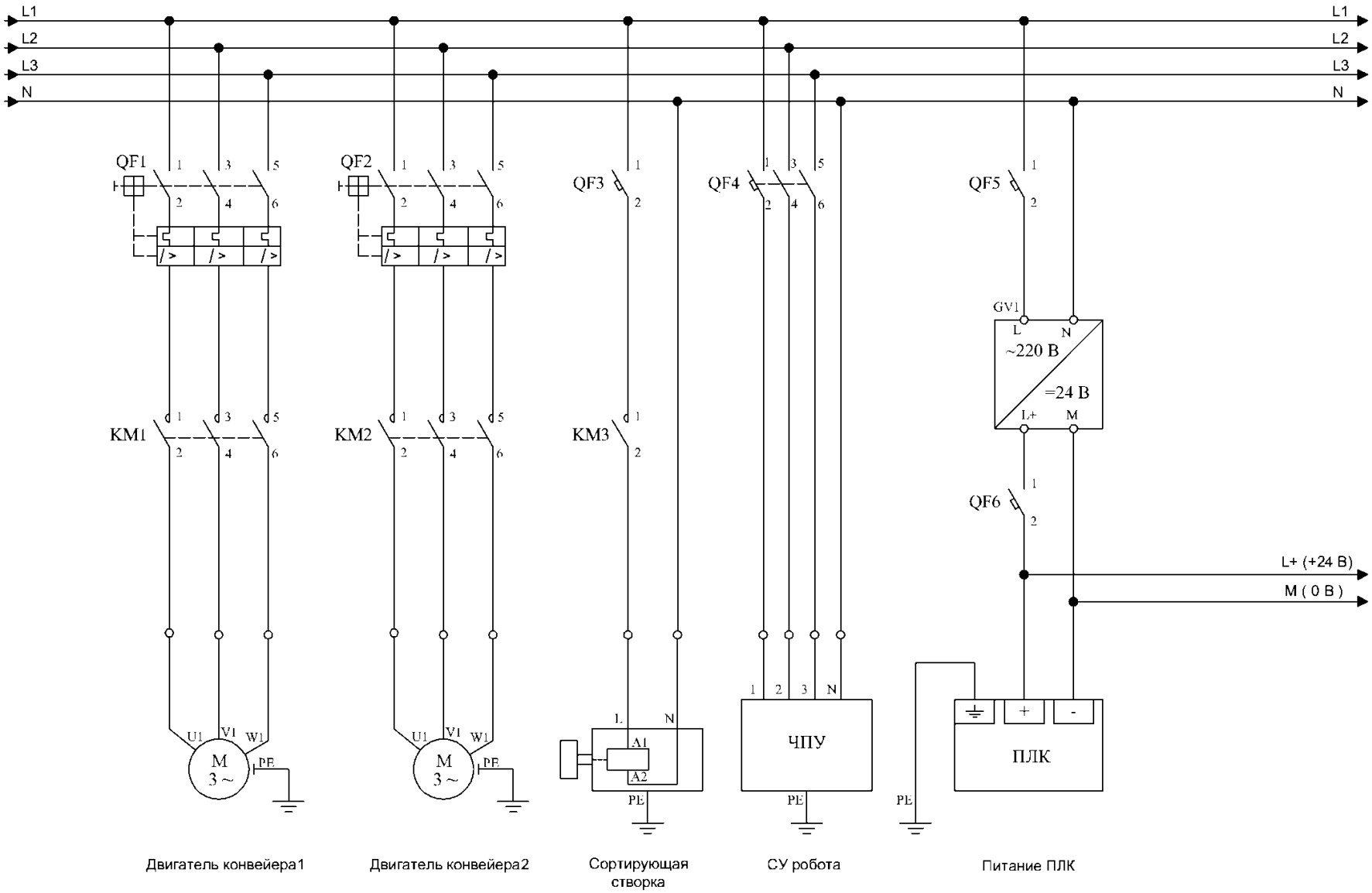



Рис. 6.6. Схема подключения оборудования (упражнение Е-2)

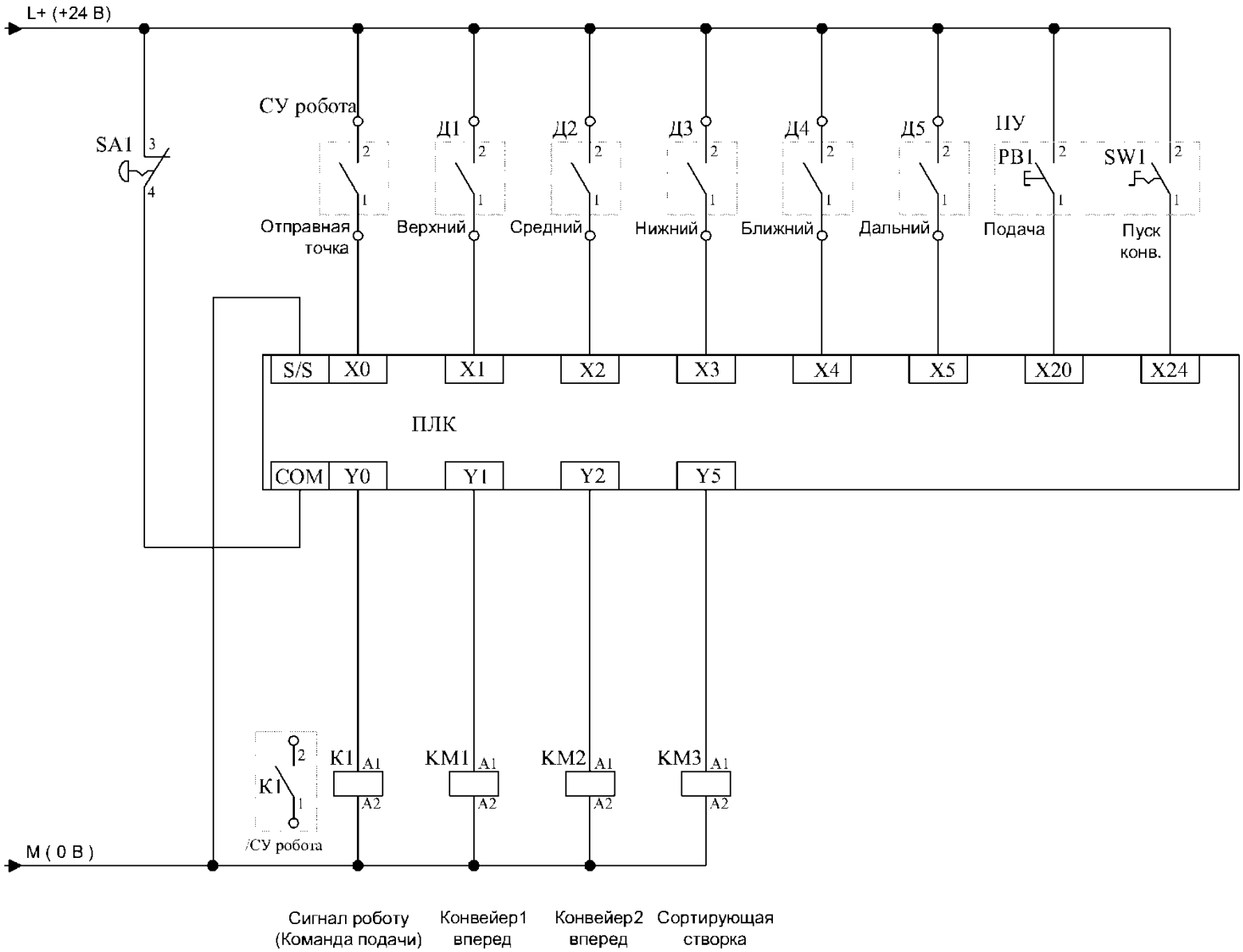



Рис. 6.7. Схема подключения входов-выходов ПЛК (упражнение Е-2)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматический выключатель для двигателя с тепловым реле
QF3…QF6 – автоматические выключатели
KM1 – контактор для управления двигателем конвейера1
KM2 – контактор для управления двигателем конвейера2
KM3 – контактор для управления сортировочной створкой
K1 – реле для подачи управляющего сигнала роботу
Д1, Д2, Д3 – датчики определения размеров детали
Д4, Д5 – датчики положения детали
PB1 – кнопка на панели управления
SW1 – тумблер на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Приведите пример использования таймера с накоплением.
2. Нарисуйте схему и временную диаграмму работы таймера с накоплением.
3. Приведите пример использования таймера с памятью.
4. Нарисуйте схемы и временные диаграммы работы таймера с памятью.
5. Из чего складывается таймер-погрешность по входу и таймер-погрешность по выходу?
6. Измените созданную в упражнении Е-2 программу таким образом, чтобы конвейеры *Y1* и *Y2* останавливались в случае, когда тыловой или передний поддон будут наполнены большими или маленькими деталями соответственно. Вместимость тылового поддона 4 детали, переднего – 6 деталей).

# ЛАБОРАТОРНАЯ РАБОТА №7

## Управление подачей деталей. Управление конвейером

**Цель работы**

Самостоятельно создать программное обеспечение для заданных в упражнениях E-5 и E-6 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

1. **Упражнение E-5 – Управление подачей деталей.**

На рисунке 7.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

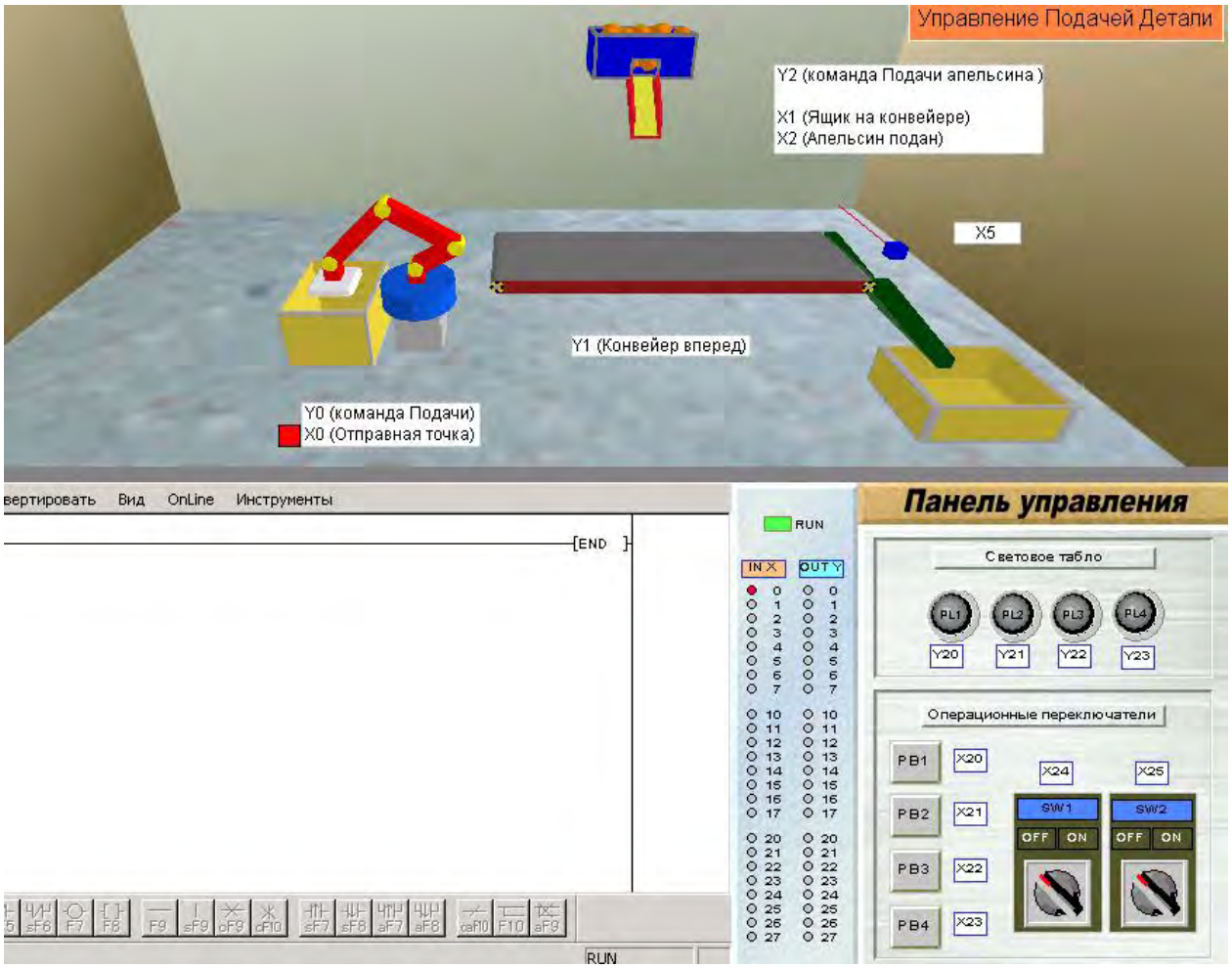



Рис. 7.1. Панель управления и виртуальное оборудование к упражнению E-5

Тумблер *SW1* на **Панели управления** (вход *X24* контроллера) управляет пуском-остановом конвейера (выход *Y1*) в положениях *ON/OFF* соответственно. Кнопка *PB1* (вход *X20* контроллера) задает для робота с выхода *Y0* управляющий сигнал подачи ящика на конвейер при условии, что робот находится в отправной точке (нормально открытый контакт входа *X0* контроллера замкнут). Датчик *X1*,

обнаружив ящик на конвейере, выдает сигнал на останов конвейера и загрузку деталей в ящик (*Y2*). Вместимость ящика 5 деталей. Подсчет поданных деталей осуществляется по входу датчика *X2*. После того, как ящик наполнен, конвейер перемещает его к поддону.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 7.2

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведена на рисунке 7.3, 7.4.

**Задание**

1.1. Разработать управляющую программу для реализации поставленной задачи подачи точно установленного числа деталей в ящик, расположенный на конвейере, с учетом следующих условий:

Общее управление

a) Когда на **Панели управления** тумблер *SW1 (X24)* переведен в *ON*, конвейер движется. Когда тумблер *SW1 (X24)* установлен в *OFF*, конвейер останавливается.
b) Когда на **Панели управления** нажата кнопка *PB1 (X20)*, роботу выдается *ON команда Подачи (Y0)*. *Команда Подачи (Y0)* переходит в *OFF*, когда робот уходит от отправной точки *(X0)*. Когда *команда Подачи (Y0)* переходит в *ON*, робот подает ящик на конвейер.

Управление деталями

a) Когда датчик для определения *Ящик на конвейере (X1)* в устройстве линии подачи деталей перейдет в *ON*, конвейер остановится.
b) В ящик помещаются пять деталей. Ящики, содержащие по 5 деталей, переносятся к поддону справа.
c) Подача деталей осуществляется, когда *Команда подать детали (Y2)* установлена в *ON* и ведется подсчет поданных деталей по установке в *ON* входа *Поданные детали (X2)*.

\

ПРЕДОСТЕРЕЖЕНИЕ

Для корректного выполнения программы необходимо задать условие, что только одна деталь может одновременно находиться на конвейере.

1.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы произвести по следующим пунктам:

a) На **Панели управления** установите в *ON* тумблер *SW1 (X24)*
Результат ► конвейер перемещается вправо.

b) На **Панели управления** нажните кнопку *PBI (X20)*
Результат ►робот подает ящик на конвейер.

c) Функционирование подачи деталей
Результат ► ящик останавливается под линией подачи и в него попадает точно установленное число деталей.

d) После того, как детали поданы
Результат ► конвейер движется вправо и переносит ящик к поддону.

e) Повторение операций
Результат ►когда нажата кнопка *PBI (X20),* осуществляется повтор функционирования, начиная с пункта b).
Для того чтобы иметь возможность повторить функционирование, следует инициализировать экран посредством щелчка на дистанционном управлении по кнопке *Reset*.

1.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению Е-5

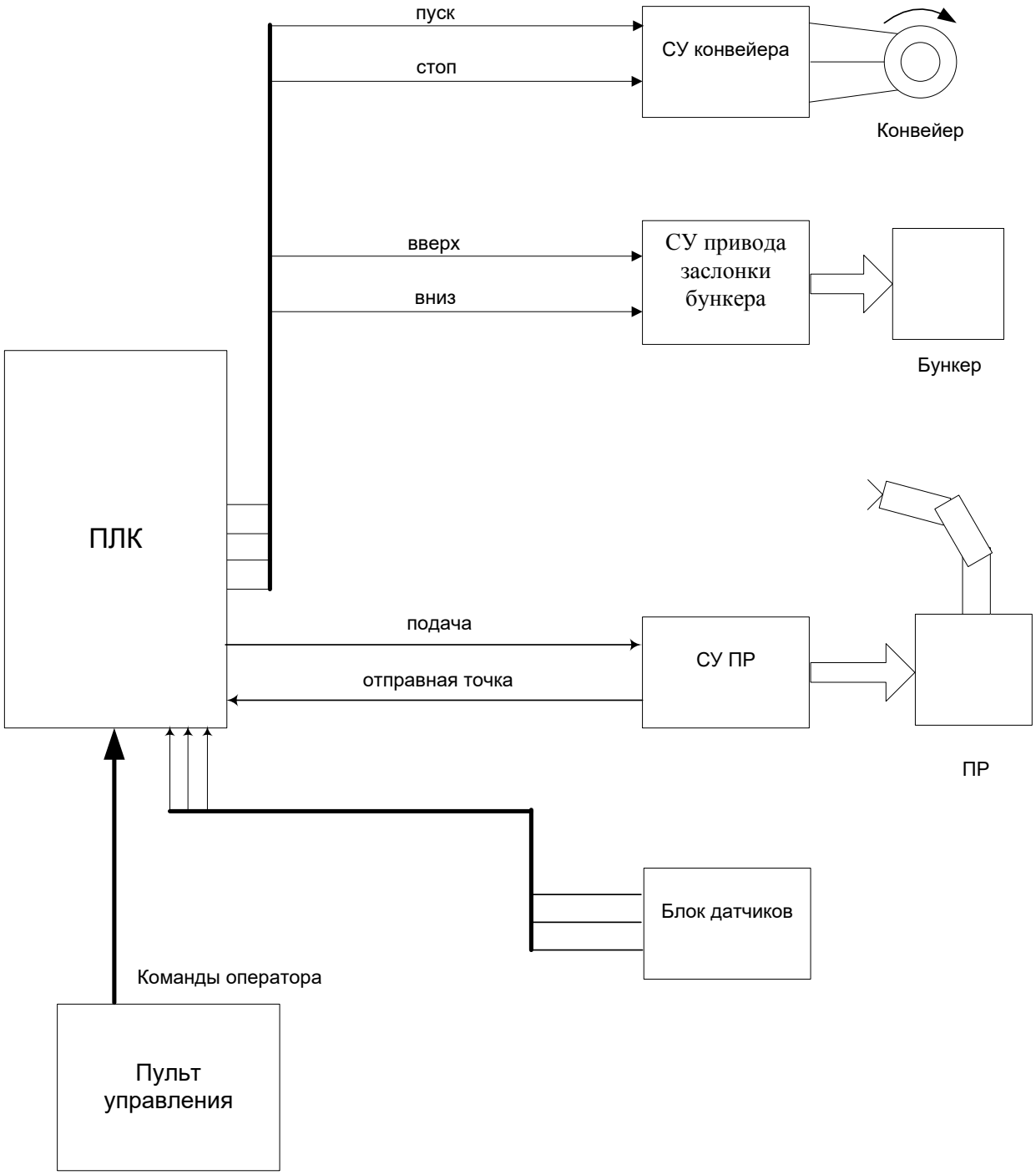



Рис. 7.2. Структурная схема СУ к упражнению Е-5

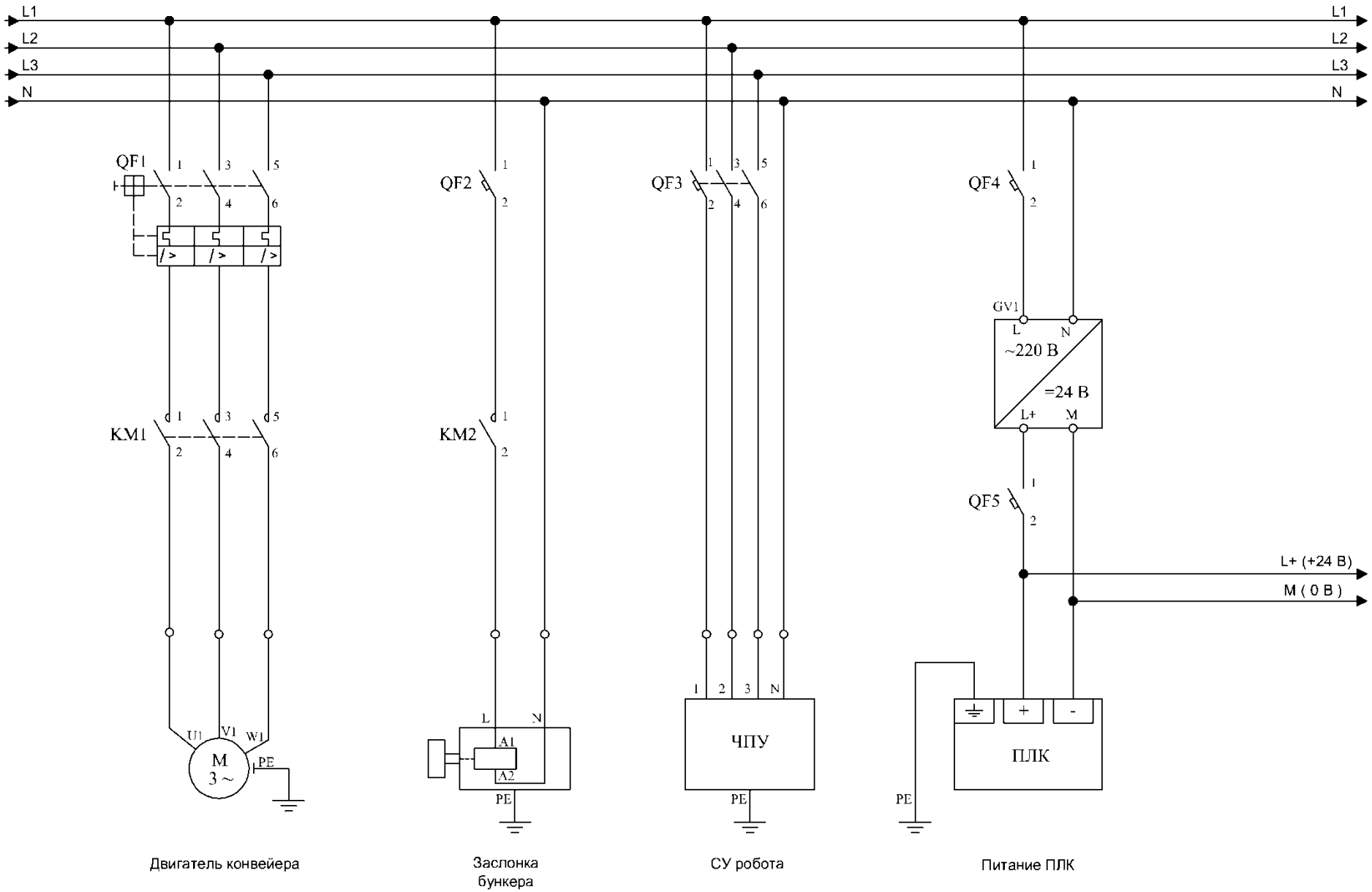



Рис. 7.3. Схема подключения оборудования (упражнение Е-5)

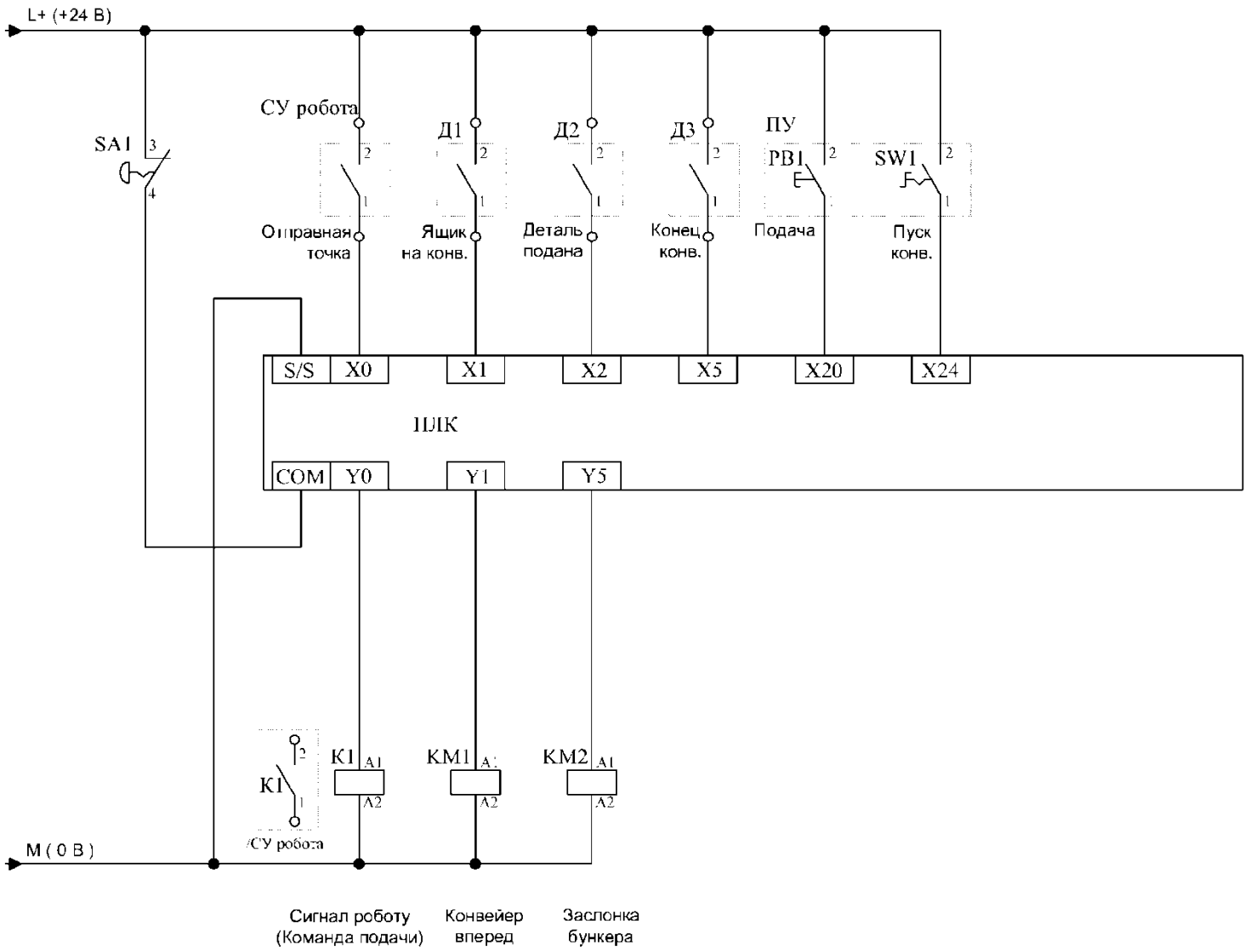



Рис. 7.4. Схема подключения входов-выходов ПЛК (упражнение Е-5)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF5 – автоматические выключатели
KM1 – контактор для управления двигателем конвейера
KM2 – контактор для управления сортирующей створкой
K1 – реле для подачи управляющего сигнала роботу
Д1…Д3 – датчики положения детали
PB1 – кнопка на панели управления
SW1 – тумблер на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

2. **Упражнение E-6 – Управление конвейером.**

На рисунке 7.4 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

Кнопка *PB1* (вход *X20* контроллера) задает для загрузочной воронки с выхода *Y10* контроллера управляющий сигнал подачи детали на конвейер. При нажатии на кнопку *PB2* (*X21)* конвейер перемещает деталь вправо *(Y11)* до тех пор, пока не сработает конечный выключатель *Правосторонний предел (X11)*, затем начинается реверсивное движение конвейера *(Y12),* которое ограничивается срабатыванием конечного выключателя *Левосторонний предел (X10).* Деталь находится в крайнем левом положении в течение 5с, после чего конвейер перемещает ее вправо, пока не сработает датчик *Останова (X12),* управляющий сигнал от которого останавливает конвейер.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 7.5.

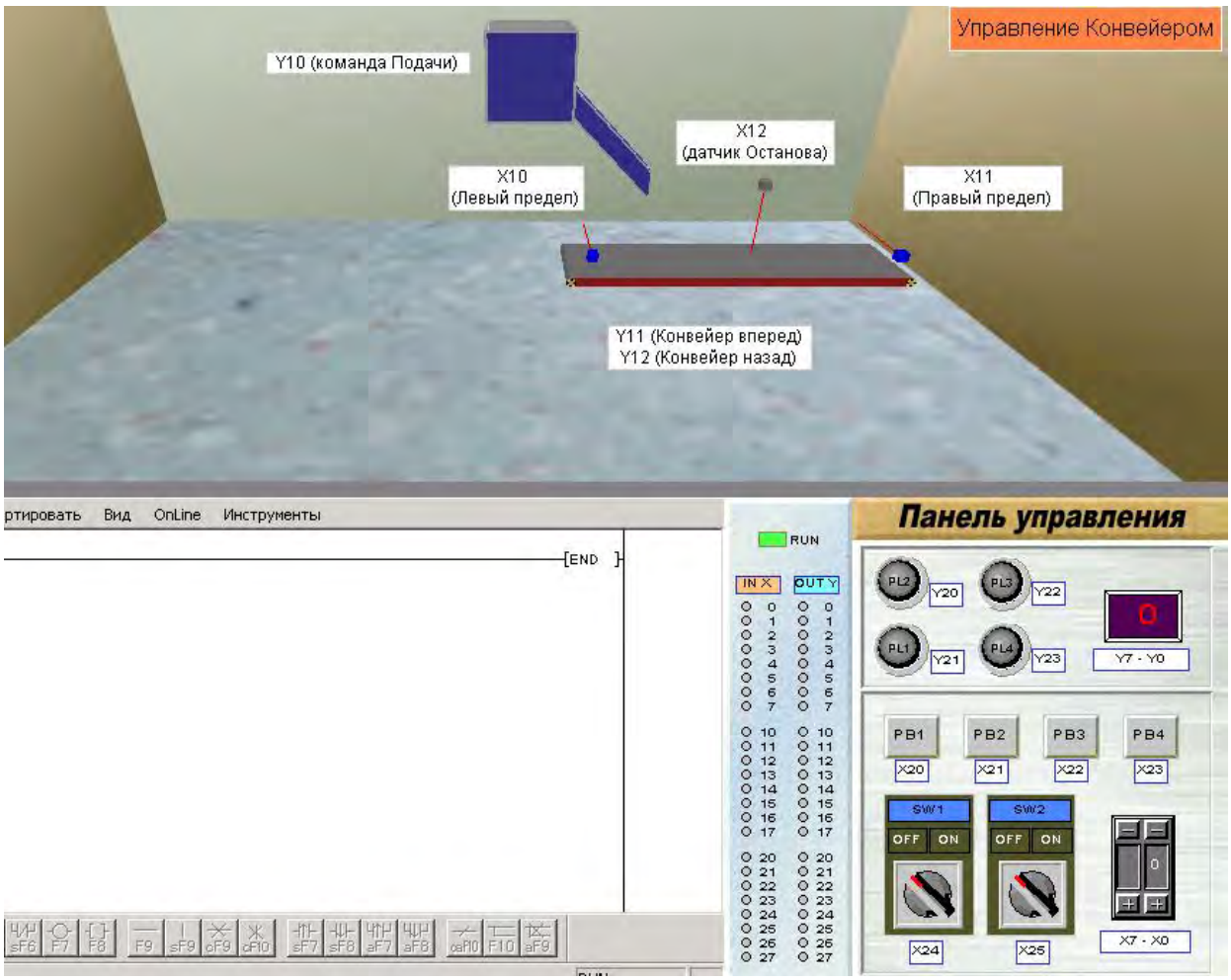



Рис. 7.4. Панель управления и виртуальное оборудование к упражнению E-6

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков Д1 реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейер приводится в движение трехфазным асинхронным двигателем с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 7.6, 7.7.

**Задание**

2.1. Разработать управляющую программу для реализации поставленной задачи управления движением конвейера с учетом следующих условий:

a) Когда на **Панели управления** нажата кнопка *PB1 (X20)*, команда *Подачи (Y10)* для загрузочной воронки переходит *ON* и загрузочная воронка

подает деталь. Когда кнопка *PBI (X20)* отпущена, команда *Подачи (Y10)* переходит в *OFF*.

b) Когда на **Панели управления** нажата кнопка *PB2 (X21),* конвейер функционирует согласно описанной ниже в пунктах с c) по f) последовательности. Если *PB2 (X21)* отпущена, последовательность функционирования продолжается.

c) При нажатии на кнопку *PB2* (*X21)* конвейер начинает перемещаться вправо *(Y11)* и останавливается, когда срабатывает конечный выключатель *Правосторонний предел (X11)*.

d) После срабатывания конечного выключателя *X11* конвейер перемещается в обратном направлении *(Y12)* до тех пор, пока не сработает конечный выключатель *Левосторонний предел (XI0)*.

e) Деталь остается у левостороннего предела в течение 5 секунд.

f) *5* секунд спустя конвейер начинает перемещаться вперед *(Y11)* до тех пор, пока не сработает датчик *Останова (X12)*.

2.2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.** Подтверждение соответствия программы провести по следующим пунктам:

a) На **Панели управления** нажмите кнопку *PB1 (X20)*
Результат ► деталь поступает из загрузочной воронки.

b) На **Панели управления** нажмите кнопку *PB2 (X21)*
Результат ►конвейер перемещается вправо и останавливается у *Правостороннего предела (X11)*.

c) Функционирование в обратном направлении
Результат ► после остановки детали у правого предела, конвейер движется влево, и деталь останавливается у *Левостороннего предела (X10)*.

d) Функционирование у левостороннего предела
Результат ► деталь остается у *Левостороннего предела (X10)* в течение *5* секунд.

e) Операция останова
Результат ► После остановки детали у левостороннего предела в течение *5* секунд, конвейер движется вправо и деталь останавливается у датчика *Останова (X12)*.
Для того чтобы иметь возможность повторить функционирование, следует инициализировать экран посредством щелчка на дистанционном управлении по кнопке *Reset*.

2.3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению Е-6

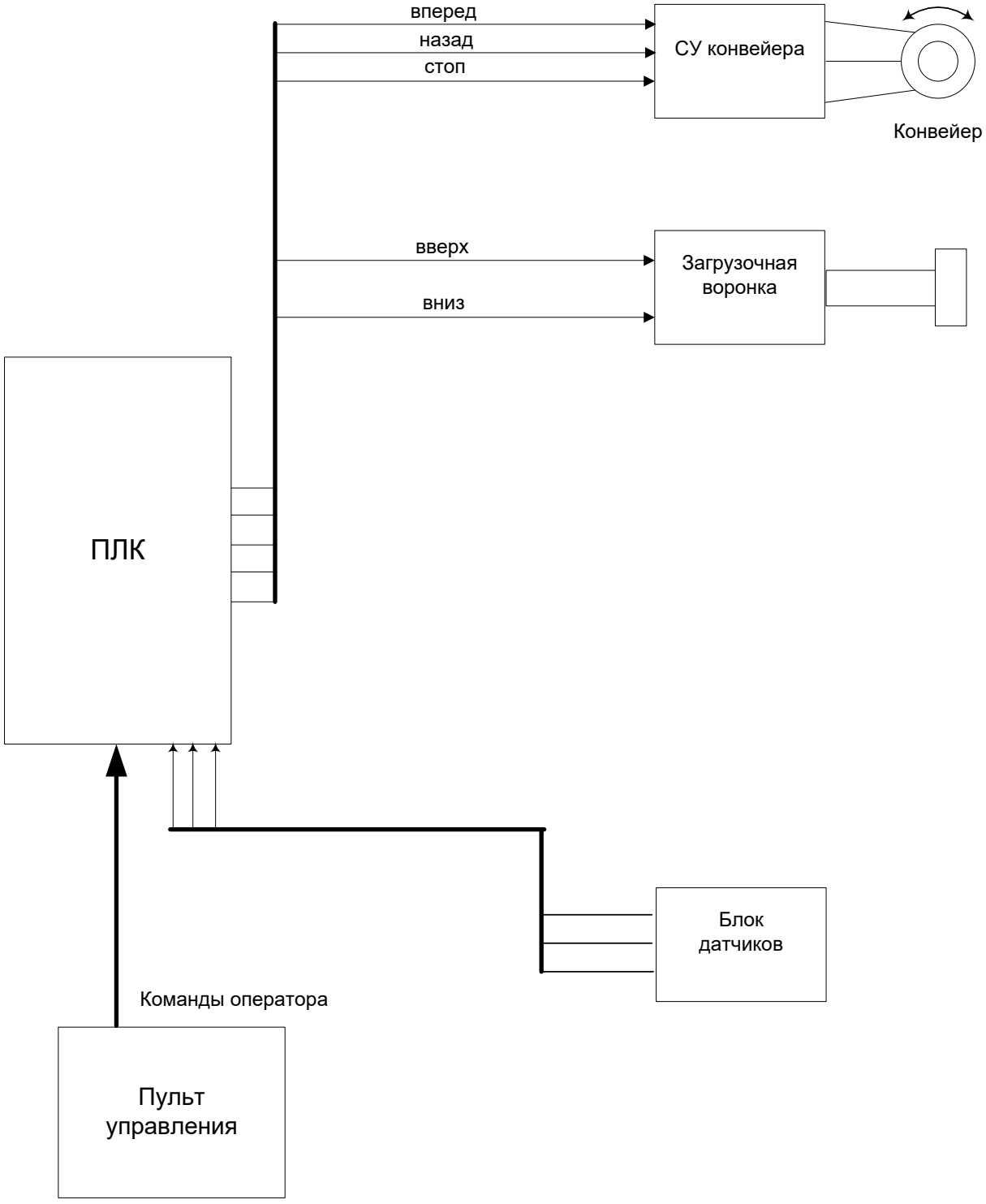



Рис. 7.5. Структурная схема СУ к упражнению Е-6

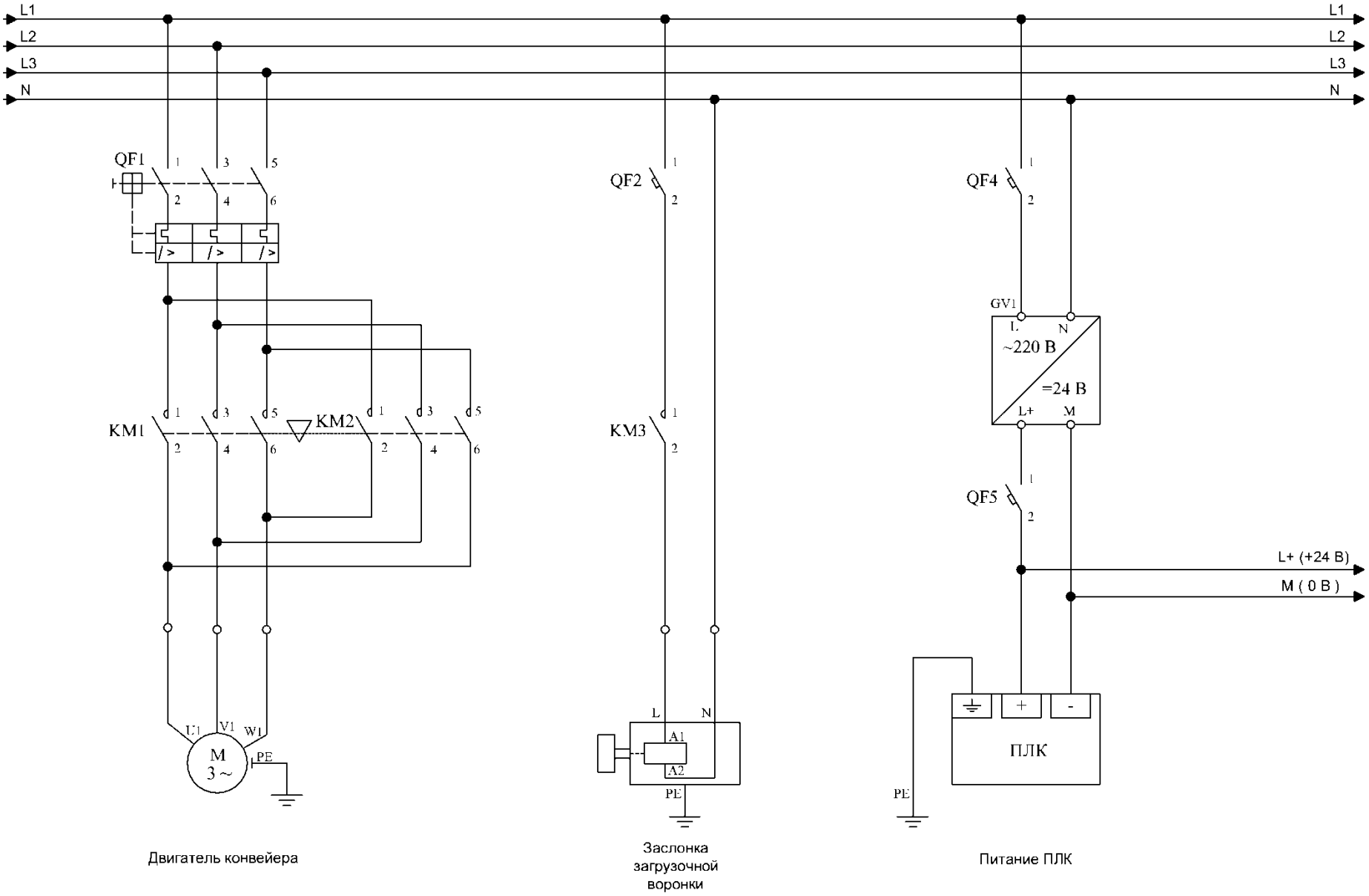



Рис. 7.6. Схема подключения оборудования (упражнение Е-6)

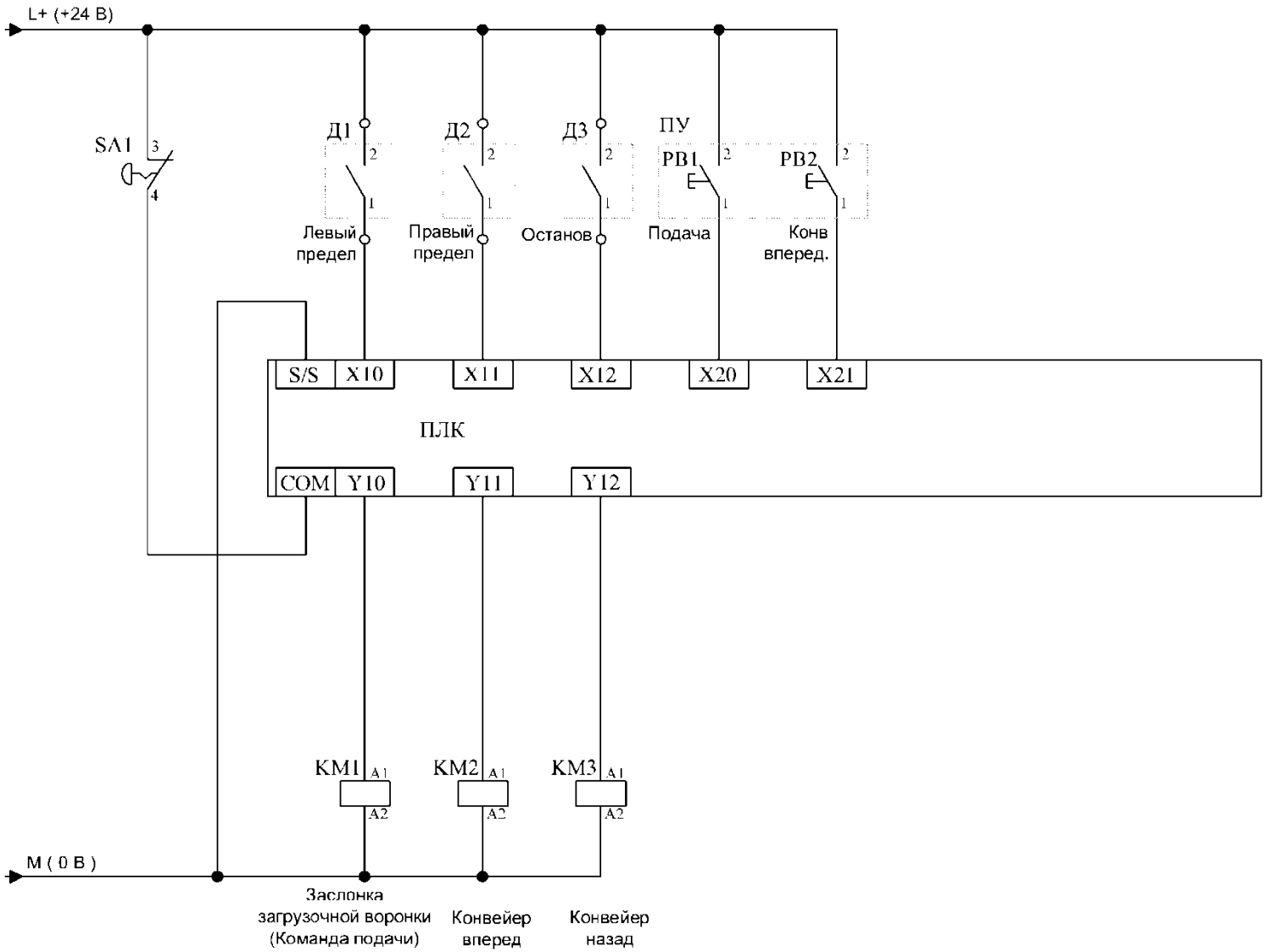



Рис. 7.7. Схема подключения входов-выходов ПЛК (упражнение Е-6)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF5 – автоматические выключатели
KM1 – контактор для управления загрузочной воронкой
KM2 – контактор для управления двигателем (конвейер вперед)
KM3 – контактор для управления двигателем (конвейер назад)
Д1…Д3 – датчики положения детали
PB1, PB2 – кнопка на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Перечислите критерии оценки промышленных сетей как средства транспортировки данных.
2. Назначение интерфейса PROFIBUS-DP. Какую скорость передачи данных он обеспечивает?
3. Опишите задачи, решаемые промышленными сетями уровня Field Level.
4. Опишите задачи, решаемые промышленными сетями уровня Sensor-Actuator Level.
5. Какие средства визуализации ТП Вы знаете?

# ЛАБОРАТОРНАЯ РАБОТА №8

## Автоматическое функционирование двери

**Цель работы**

Самостоятельно создать программное обеспечение для заданной в упражнении F-1 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

**Упражнение F-1 – Автоматическое функционирование двери.**

На рисунке 8.1 показано виртуальное оборудование: дверь, которая поднимается вверх и опускается вниз по сигналам от датчиков *У ворот (X2)* и *За воротами (X3)* соответственно, *Устройство звуковой сигнализации (Y7)*, которое работает во время движения двери, и *Лампа (Y6)*, которая включается, когда автомобиль находится между датчиками *У ворот (X2)* и *За воротами (X3)*. Три лампы на **Панели управления** служат для индикации состояния двери: *Дверь в движении (Y11), Дверь закрыта (Y10), Дверь открыта (Y13)*. Четвертая лампа *Свет у ворот (Y10)* горит, когда горит лампа *(Y6)*. Дверь может быть открыта и закрыта вручную при нажатии кнопок *Дверь вверх (X10)* и *Дверь вниз (X11)* на **Панели управления**.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 8.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление виртуальным оборудованием. Блок датчиков регистрирует местонахождение автомобиля, выходные сигналы датчиков поступают на входы контроллера, программно обрабатываются, и с выходов ПЛК осуществляется управление открытием-закрытием двери, устройством сигнализации и включением-выключением лампы над дверью.

Схема подключения оборудования и входов-выходов ПЛК приведена на рисунке 8.3.

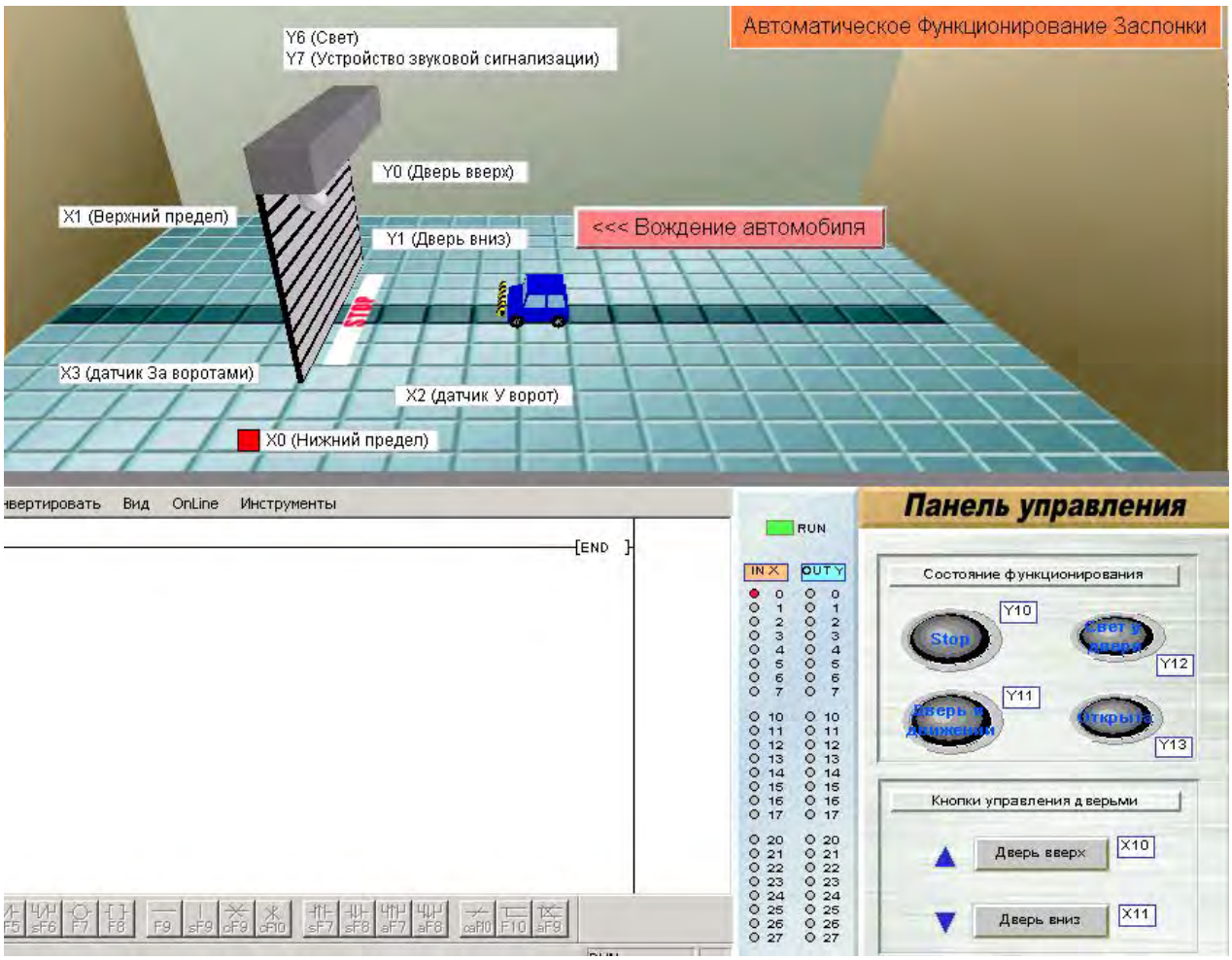



Рис. 8.1. Панель управления и виртуальное оборудование к упражнению F-1

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи управления функционированием двери с учетом следующих условий:
   a) Когда датчик *У ворот (X2)* обнаруживает автомобиль, дверной механизм поднимает ее вверх.
   b) Когда автомобиль проезжает сквозь дверной проем, дверь опускается.
   c) При движении вверх дверь останавливается, когда срабатывает конечный выключатель *Верхний предел (X1)*.
   d) При движении вниз дверь останавливается, когда срабатывает конечный выключатель *Нижний предел (X0)*.
   e) Пока автомобиль находится между датчиком *У ворот (X2)* и датчиком *За воротами (X3),* дверь не опускается.
   f) *Устройство звуковой сигнализации (У7)* функционирует во время движения двери.
   g) *Лампа (У6)* горит во время нахождения автомобиля между *датчиком У ворот (X2)* и *датчиком За воротами (X3)*.
   h) Три лампы на **Панели управления** служат для индикации состояния двери: *Дверь в движении (Y11), Дверь не движется – STOP (Y10), Дверь*

*открыта (Y13).* Четвертая лампа *Свет у ворот (Y12)* горит, когда горит лампа *(Y6).*

i) Дверь может быть открыта или закрыта вручную посредством нажатия на **Панели управления** соответствующих кнопок ▲ *Дверь вверх (X10)* и ▼ *Дверь вниз (X11).*

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации.** Подтверждение соответствия программы произвести по следующим пунктам:

a) Для того, чтобы переместить автомобиль к двери, нажмите кнопку *Управление автомобилем*
Результат ► дверь поднимается вверх, когда *датчик У Ворот (X2)* обнаруживает автомобиль.

b) Работа *Устройства звуковой сигнализации*
Результат ► во время движения двери устройство звуковой сигнализации издает тональные звуки. Устройство звуковой сигнализации отключается, если дверь остановлена. (Если в вашем компьютере отсутствует звуковая карта, то проверьте *ON/OFF* состояние выхода (*Y7*).

c) Свет над дверьми
Результат ►В момент нахождения автомобиля между датчиком *У ворот (X2)* и датчиком *За воротами (X3)* горит *Лампа желтого цвета*.

d) Нажмите кнопку *Управление автомобилем* и позвольте автомобилю проехать сквозь ворота
Результат ►Когда *датчик За воротами (X3)* переходит в состояние *OFF*, дверь движется вниз.

e) Операции, выполняемые на **Панели управления** вручную
Результат ►Дверь может быть вручную открыта и закрыта за счет нажатия кнопок ▲ *Дверь вверх (X10)* и ▼ *Дверь вниз (X11).* Однако, в то время как автомобиль находится в диапазоне обнаружения (между датчиками), дверь не может быть опущена.

f) Лампы индикации на **Панели управления**
Результат ►Каждая лампа загорается и гаснет согласно перемещению двери и автомобиля.
Для того чтобы иметь возможность повторить функционирование, следует инициализировать экран посредством щелчка на дистанционном управлении по кнопке *Reset*.

3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-1

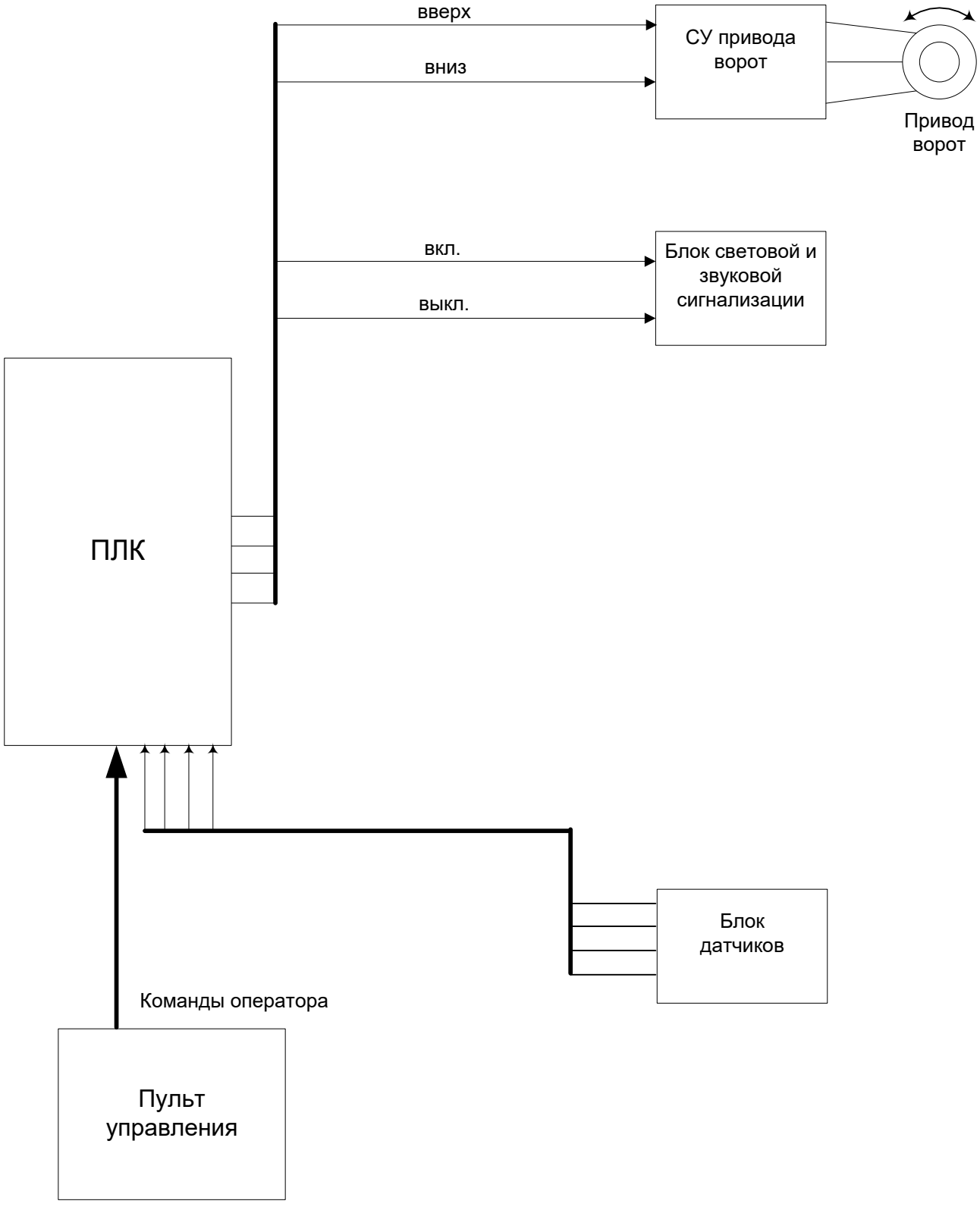



Рис. 8.2. Структурная схема СУ к упражнению F-1

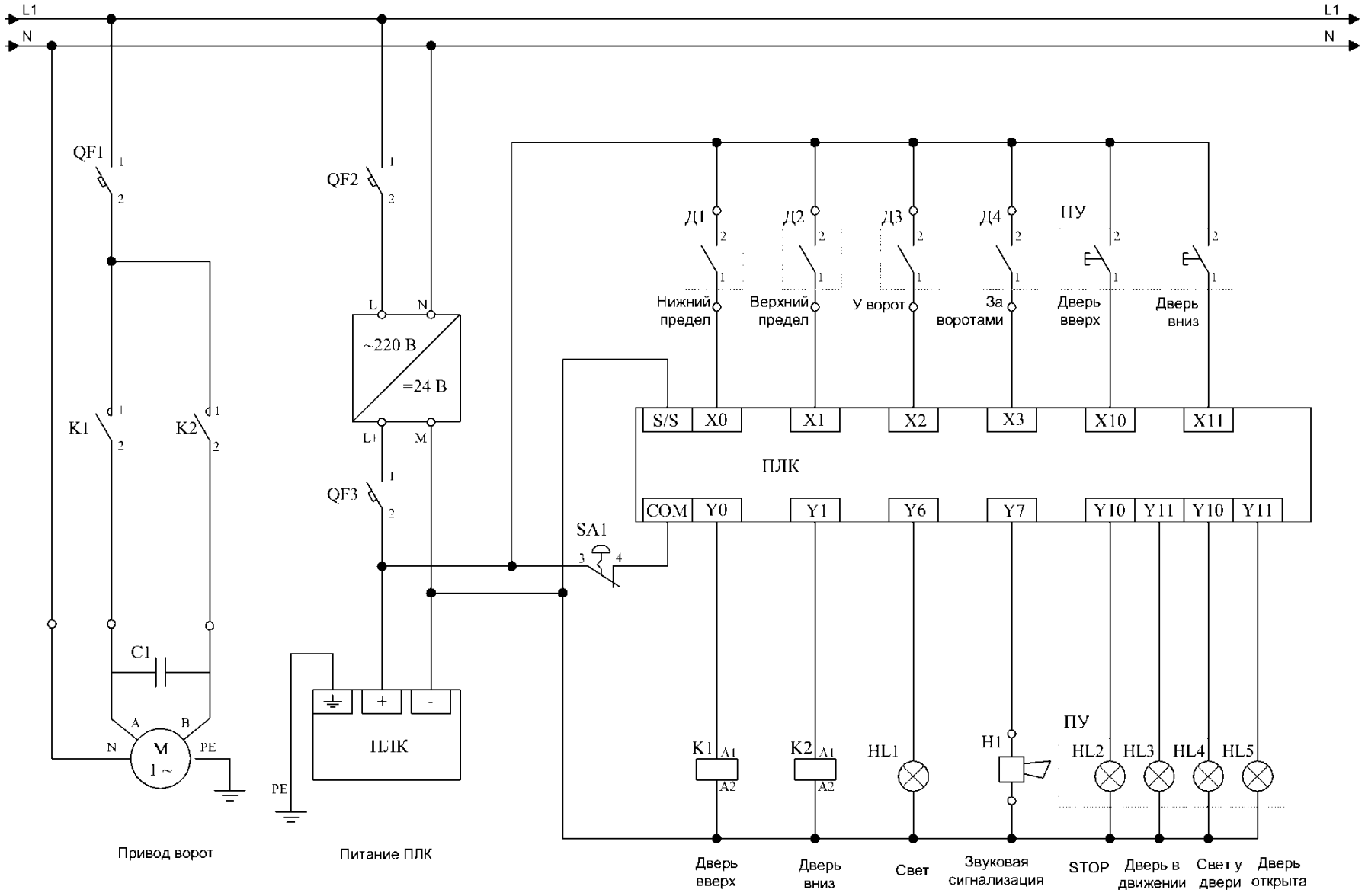



Рис. 8.3. Схема подключения СУ (упражнение F-1)

L1 – однофазный источник питания (220В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2, QF3 – автоматические выключатели
K1 – реле управления двигателем (поднять ворота)
K2 – реле управления двигателем (опустить ворота)
Д1…Д4 – датчики положения
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
HL1…HL5 – лампы световой индикации
H1 – устройство звуковой сигнализации

**Контрольные вопросы**

1. Перечислите основные электрические параметры ПЛК.
2. Перечислите основные программные параметры ПЛК.
3. Какие устройства непосредственно подключаются к дискретным входам ПЛК?
4. Почему для аналого-цифрового преобразования в ПЛК используются АЦП разрядностью не выше 8÷12?
5. В каких случаях возникает необходимость использования специализированных входов вместо стандартных дискретных?
6. Какие варианты исполнения выходов ПЛК Вы знаете? В чем их основные достоинства и недостатки?
7. Какие выходы, дискретные или аналоговые, более надежные и почему?

# ЛАБОРАТОРНАЯ РАБОТА №9

## Управление виртуальным оборудованием сцены

**Цель работы**

Самостоятельно создать программное обеспечение для заданной в упражнении F-2 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

**Упражнение F-2 – Управление виртуальным оборудованием сцены.**

На рисунке 9.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками, с помощью которых задаются сигналы управления виртуальным оборудованием сцены.

Кнопка *Начать* (вход *X16* контроллера) на **Панели управления** при условии, что занавес закрыт и сцена расположена у *Нижнего предела (X7)*, задает для *Устройства звуковой сигнализации* с выхода *Y5* управляющий сигнал включения. *Устройство звуковой сигнализации* (*Y5*) работает в течение 5с, после чего выдается команда *Открытия занавеса (Y0)*. Положение левого занавеса контролируется концевыми выключателями *Левый занавес закрыт (X0)*, *Левый занавес в среднем положении (X1), Левый занавес открыт (X2)*. Положение правого занавеса контролируется концевыми выключателями *Правый занавес закрыт (X3)*, *Правый занавес в среднем положении (X4), Правый занавес открыт (X5)*. После срабатывания конечных выключателей *Левый занавес открыт (X2)* и *Правый занавес открыт (X5)* выдается команда на *Подъем сцены (Y2)* до уровня, контролируемого конечным выключателем *Верхний предел сцены (X6).* При нажатии на кнопку *Завершить* (*X17*) выдается команда *Закрытия занавеса (Y1)* и левая и правая половины занавеса закрываются до тех пор, пока не сработают концевые выключатели *Левый занавес закрыт (X0)* и *Правый занавес закрыт (X3)*. При ручном управлении команды *Открытия/Закрытия занавеса* задаются кнопками на **Панели управления** *X10/X11*, команды *Подъема* и *Опускания сцены* – кнопками ▲*Подьем сцены (X12)* и ▼*Опускание сцены (X13)* соответственно.


Рис. 9.1. Панель управления и виртуальная сцена к упражнению F-2

Лампы индикации на **Панели управления** (см. Рис. 9.1) загораются и гаснут в соответствии с операциями, проводимыми с занавесом и сценой.

Для реализации поставленной задачи управления виртуальным оборудованием сцены предлагается система управления, структурная схема которой показана на рисунке 9.2.

Основным управляющим элементом системы является ПЛК, который согласно состоянию опрашиваемых входов по записанной в память программе изменяет состояние выходов, т.е. реализует управление виртуальным оборудованием сцены.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 9.3, 9.4.

**Задание**

2.4. Разработать управляющую программу для реализации поставленной задачи управления виртуальным оборудованием сцены с учетом следующих условий:

Спецификация для операций, выполняемых автоматически

e) Когда на **Панели управления** нажата кнопка *Начать (X16)*, в течение 5 секунд работает *Устройство звуковой сигнализации (Y5)*. *Устройство звуковой сигнализации (Y5)* срабатывает только в том случае, если занавес закрыт и сцена расположена у *Нижнего предела (X7)*.

f) По окончании работы *Устройства звуковой сигнализации (Y5)*, команда *Открытия занавеса (Y0)* устанавливается в состояние *ON* и занавес раздвигается в обе стороны до достижения предельных положений, контролируемых посредством конечных выключателей *X2* и *X5*.

g) После полного раскрытия занавеса, начинается *Подъем сцены (Y2)* до срабатывания конечного выключателя *Верхний предел сцены (X6)*.

h) Когда на **Панели управления** нажата кнопка *Завершить* (*X17*), команда *Закрытия занавеса (Y1)* устанавливается в состояние *ON* и занавес закрывается до тех пор, пока не сработают концевые выключатели *Левый занавес закрыт (X0)* и *Правый занавес закрыт (X3)*.

Спецификация для операций, выполняемых вручную

Выполнение следующих операций допускается только в отсутствие описанных выше выполняющихся автоматических операций:

a) Занавес может быть раскрыт только при нажатой на **Панели управления** кнопке *Открытие Занавеса (X10).* Занавес останавливается, когда срабатывают конечные выключатели *Левый занавес открыт (X2)* и *Правый занавес открыт (X5).*

b) Занавес может быть закрыт только при нажатой на **Панели управления** кнопке *Закрытие Занавеса (X11)*. Занавес закрывается до срабатывания конечных выключателей *Левый занавес закрыт (X0)* и *Правый занавес закрыт (X3)*.

c) Сцена может двигаться вверх только при нажатой на **Панели управления** кнопке ▲*Подьем сцены (X12)*. Сцена поднимается до срабатывания конечного выключателя *Верхний предел сцены (X6)*.

d) Сцена может двигаться вниз только при нажатой на **Панели управления** кнопке ▼*Опускание сцены (X13)*. Сцена опускается до срабатывания конечного выключателя *Нижний предел сцены (X7)*.

e) Лампы индикации на **Панели управления** загораются или гаснут в соответствии с операциями, проводимыми с занавесом и сценой.

2.5. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:

2.1. На **Панели управления** нажмите кнопку *Начать (X16)*
Результат ►выполняются операции, приведенные ниже с пункта a) по c).
a) *Устройство звуковой сигнализации (Y5)* работает в течение 5 секунд.
b) После выключения *Устройства звуковой сигнализации (Y5)* занавес полностью раскрывается.
c) После раскрытия занавеса сцена начинает подниматься.

2.2. На **Панели управления** нажмите кнопку *Завершить* (*X17*)
Результат ►закрывается занавес.

2.3. Операции, выполняемые вручную
Результат ► Занавес может быть раскрыт и закрыт нажатием кнопок *Открытие Занавеса (X10)* и *Закрытие Занавеса (X11)* соответственно. Сцена может быть поднята или опущена при нажатии кнопок ▲*Подьем сцены (X12)* и ▼*Опускание сцены (X13).*

2.4. Лампы индикации на **Панели управления**
Результат ► лампы загораются и гаснут согласно состоянию выполняемых операций.

2.6. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-2

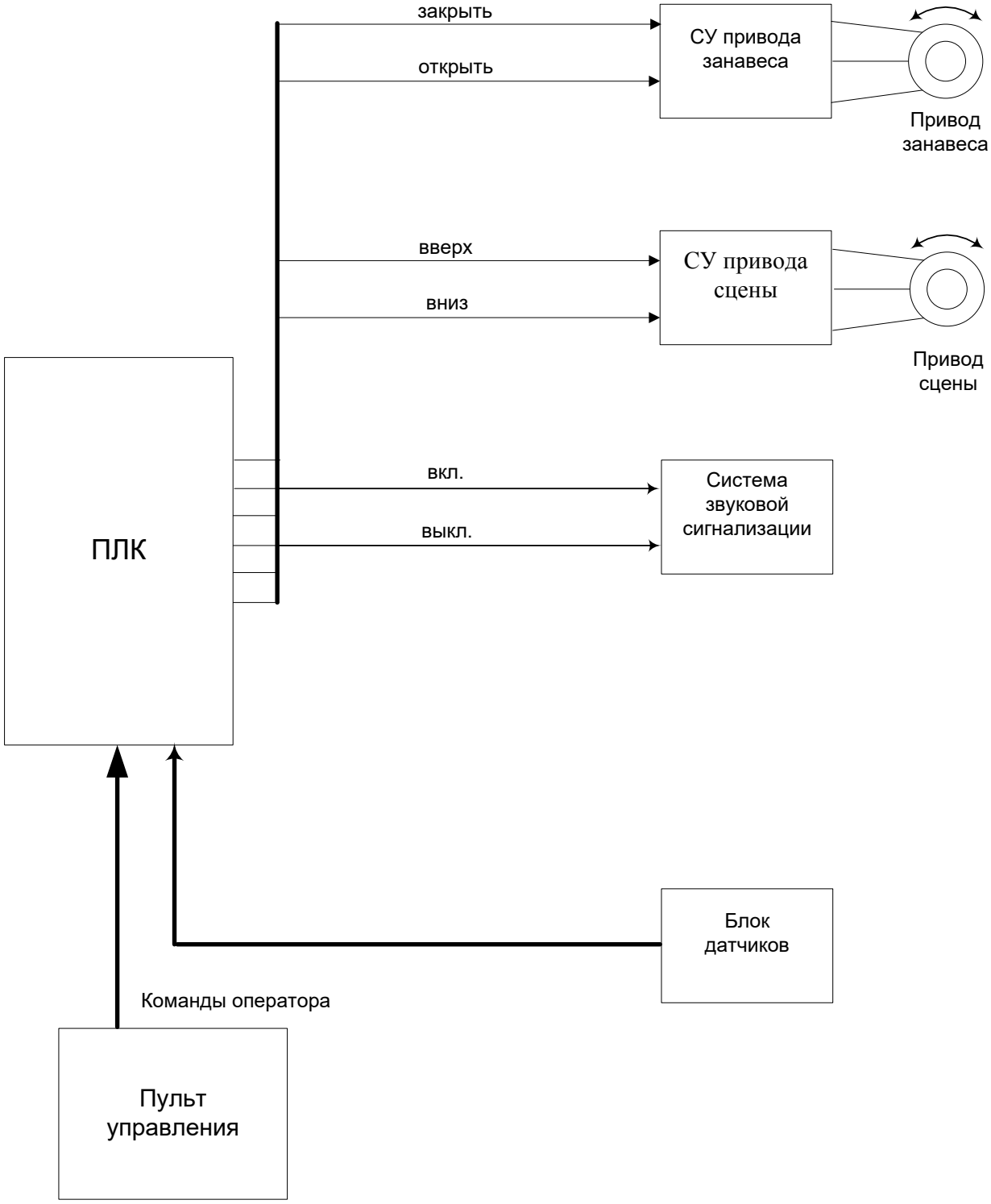



Рис. 9.2. Структурная схема СУ к упражнению F-2

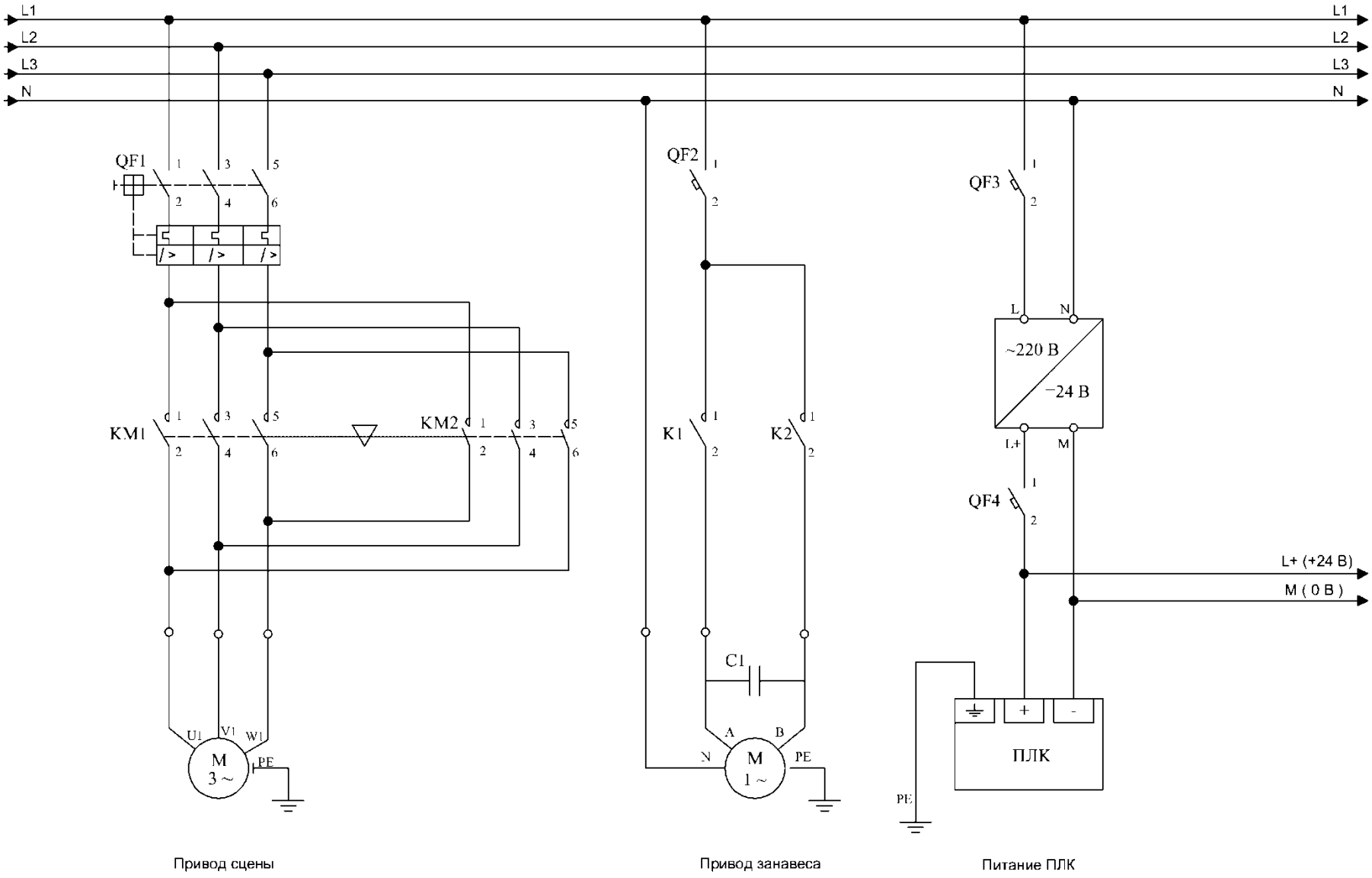



Рис. 9.3 – Схема подключения оборудования (упражнение F-2)

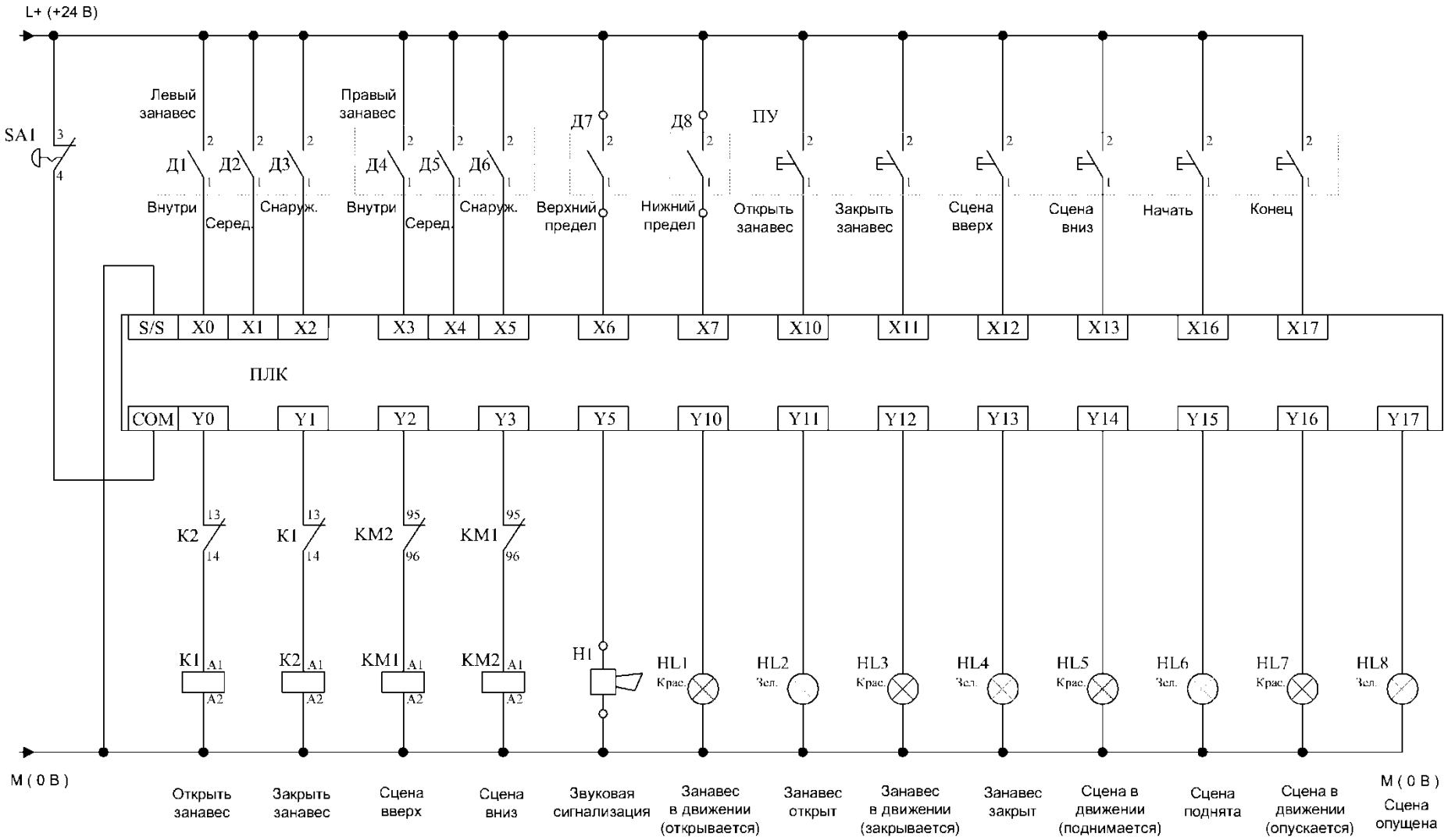



Рис. 9.4. Схема подключения входов-выходов ПЛК (упражнение F-2)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF4 – автоматические выключатели
KM1 – контактор для управления двигателем (поднять сцену)
KM2 – контактор для управления двигателем (опустить сцену)
K1 – реле для управления двигателем (открыть занавес)
K2 – реле для управления двигателем (закрыть занавес)
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
Д1…Д8 – датчики положения
HL1…HL8 – световая индикация
H1 – устройство звуковой сигнализации

**Контрольные вопросы**

1. Какими критериями оценки необходимо руководствоваться при выборе ПЛК?
2. Охарактеризуйте 4 этапа выбора ПЛК?
3. Организация памяти в контроллерах.
4. Какие параметры следует учитывать при подключении нагрузки к выходам ПЛК?
5. Покажите схематично структуру команды на языке релейно-контактных схем.

# ЛАБОРАТОРНАЯ РАБОТА №10

## Сортировка установленного числа деталей по размеру

### Цель работы

Самостоятельно создать программное обеспечение для заданной в упражнении F-3 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

### Упражнение F-3 – Сортировка установленного числа деталей по размеру.

На рисунке 10.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

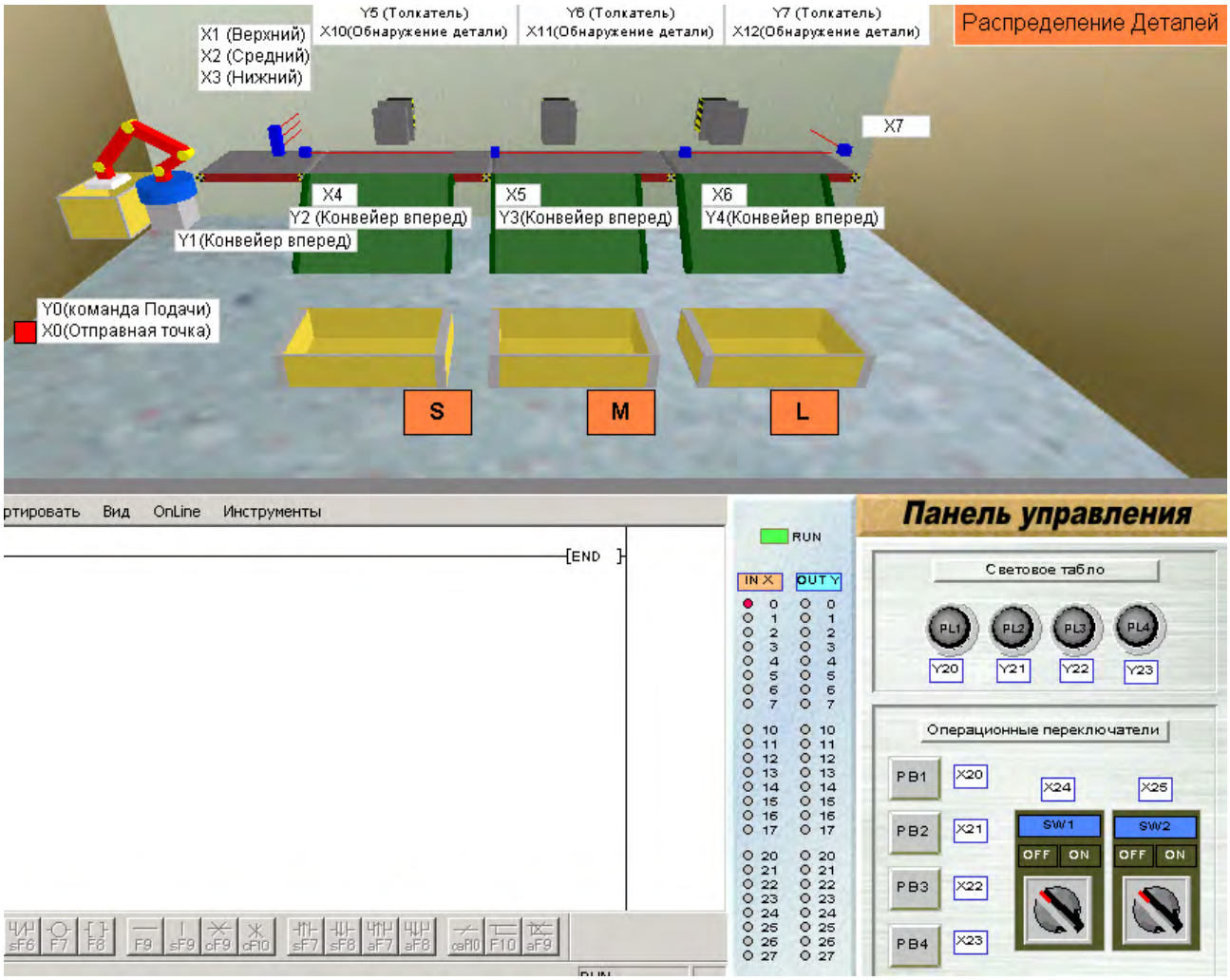



Рис. 10.1. Панель управления и виртуальное оборудование к упражнению F-3

Кнопка *PB1* (вход *X20* контроллера) на **Панели управления** в состоянии *ON* задает для робота с выхода *Y0* управляющий сигнал подачи детали на конвейер. Тумблер *X24* управляет пуском-остановом конвейеров с выходов *Y1,*

*Y2, Y3, Y4* контроллера в положениях *ON/OFF* соответственно. Датчики *X1, X2, X3* фиксируют прохождение деталей большой, средней и малой величины соответственно. Каждая деталь сталкивается одним из толкателей в определенный поддон в зависимости от ее размера: большие детали – в поддон L, средние – в поддон M, детали малой величины – в поддон S. В случае заполнения поддона детали соответствующего размера падают с правой стороны.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 10.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводится в движение трехфазными асинхронными и однофазными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 10.3, 10.4.

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи сортировки точно установленного числа деталей по размерам с учетом следующих условий:
   a) Когда на **Панели управления** нажата кнопка *PB1 (X20)*, команда *Подачи (Y0)* для робота установлена в состояние *ON*. Команда *Подачи (Y0)* переходит в состояние *OFF*, когда робот завершит перемещение детали и возвратится в отправную точку.
   b) Когда тумблер *SW1 (X24)* на **Панели управления** установлен в положение *ON*, конвейеры двигаются. Если тумблер *SW1 (X24)* установлен в положение *OFF*, конвейеры останавливаются.
   c) Детали больших, средних и малых размеров, расположенные на конвейерах, сортируются по состояниям входов датчиков Верхний (X1), Средний (X2), *Нижний (X3)* и переносятся к поддонам *L*, *M*, и *S* соответственно.
   d) Когда расположенный в толкателе датчик *Обнаружения детали (X10, X11* или *X12)* переходит в состояние *ON*, конвейер останавливается и деталь сталкивается на поддон.

a) На поддонах должно оказаться указанное число деталей каждого из размеров.
   Поддон L: 3 детали
   Поддон M: 2 детали
   Поддон S: 2 детали
   Избыточные детали должны проходить перед толкателями и падать с правой стороны.

ПРЕДОСТЕРЕЖЕНИЕ

Для корректного выполнения программы необходимо задать условие, что только одна деталь может одновременно находиться на конвейере.

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:
   a) На **Панели управления** установите в положение *ON* тумблер *SW1 (X24)*
      Результат ► конвейеры двигаются вправо.
   b) На **Панели управления** нажните кнопку *PB1 (X20)*
      Результат ► робот подает деталь.
   c) Сортировка по размеру
      Результат ►каждая деталь останавливается перед толкателем согласно ее размеру (большая, средняя или малой величины) и сталкивается на поддон.
   d) После заполнения соответствующих поддонов
      Результат ►соответствующие детали проходят перед датчиком и падают с правой стороны.
3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-3

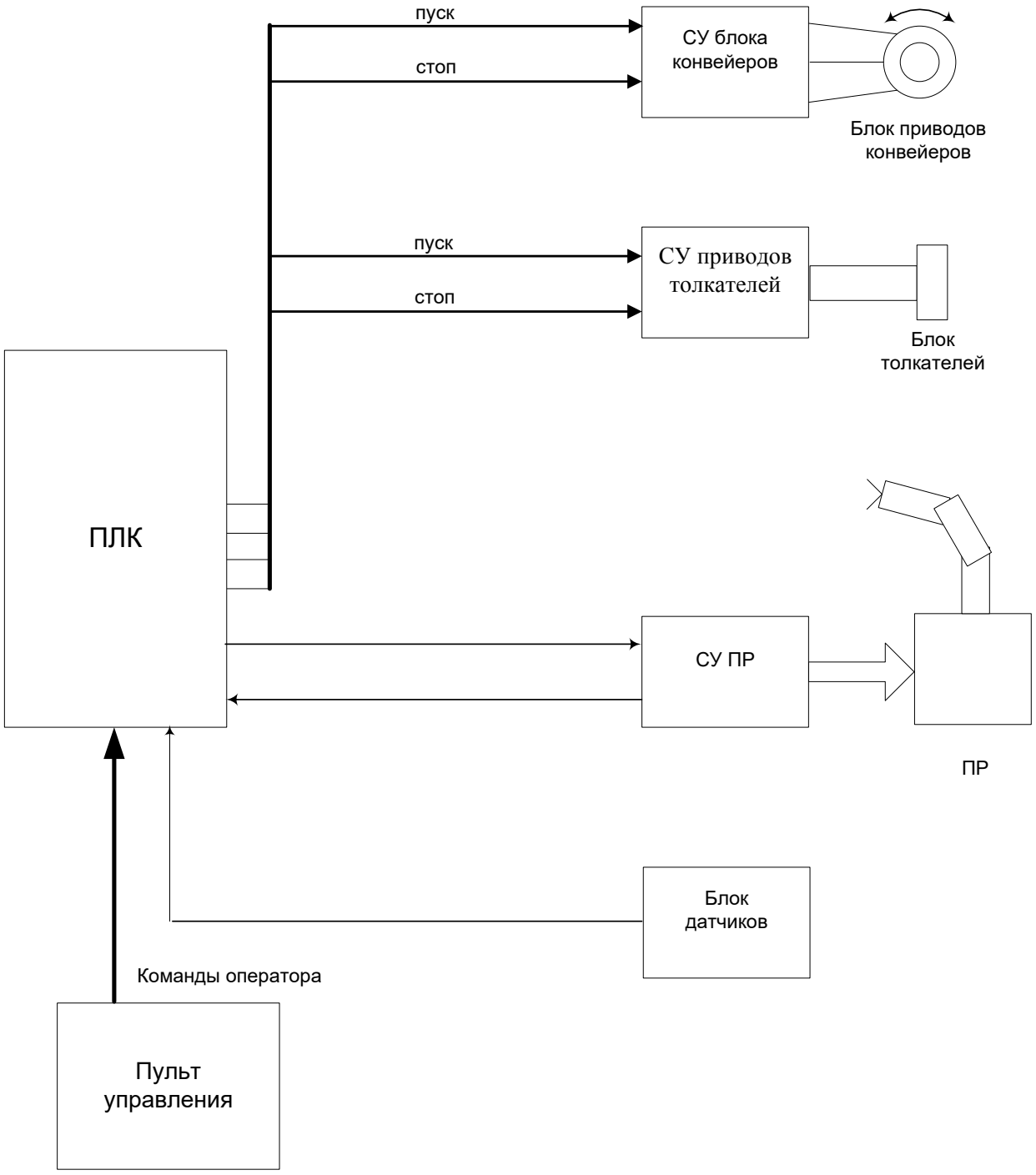



Рис. 10.2. Структурная схема СУ к упражнению F-3

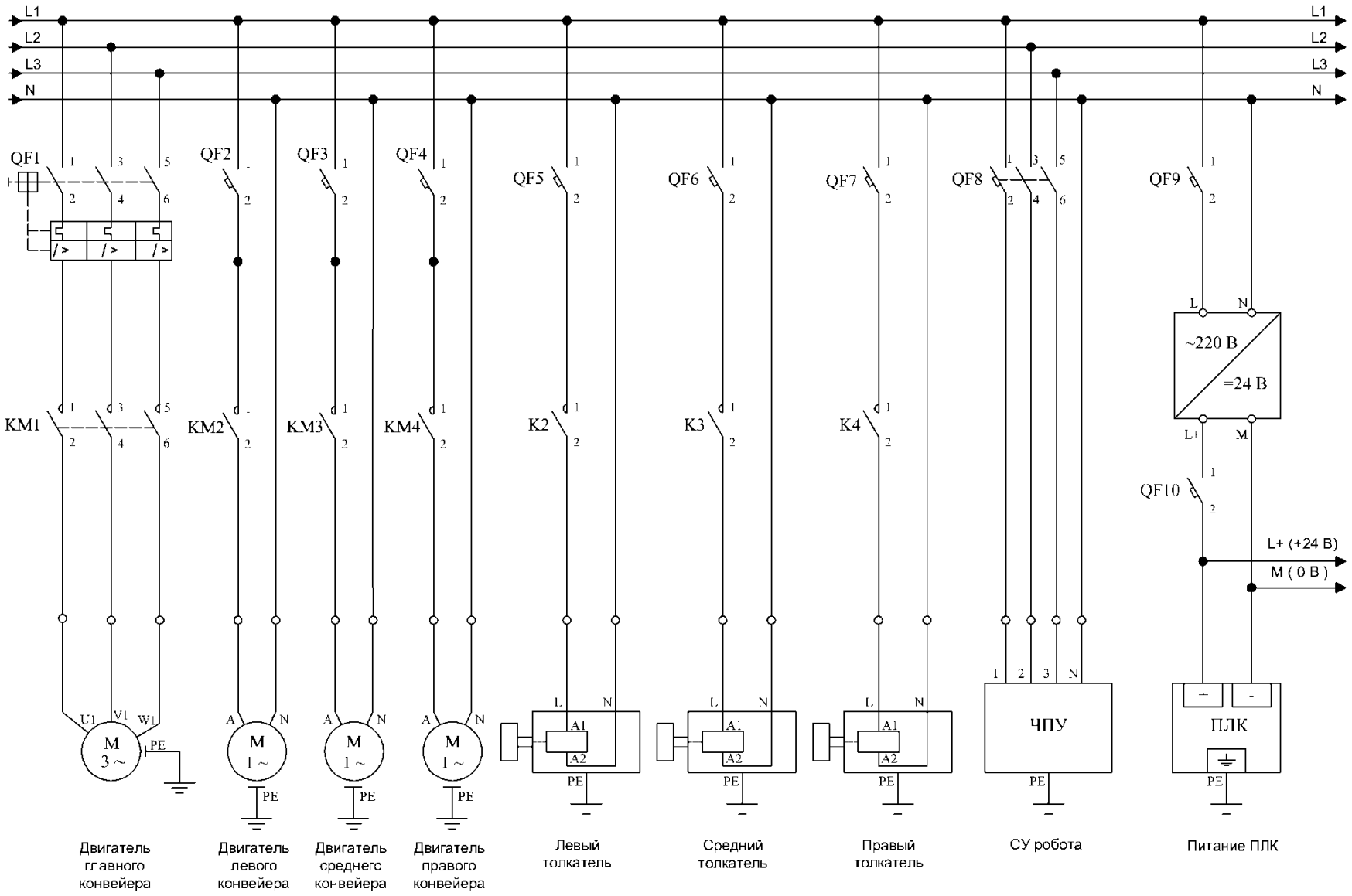



Рис. 10.3 – Схема подключения оборудования (упражнение F-3)

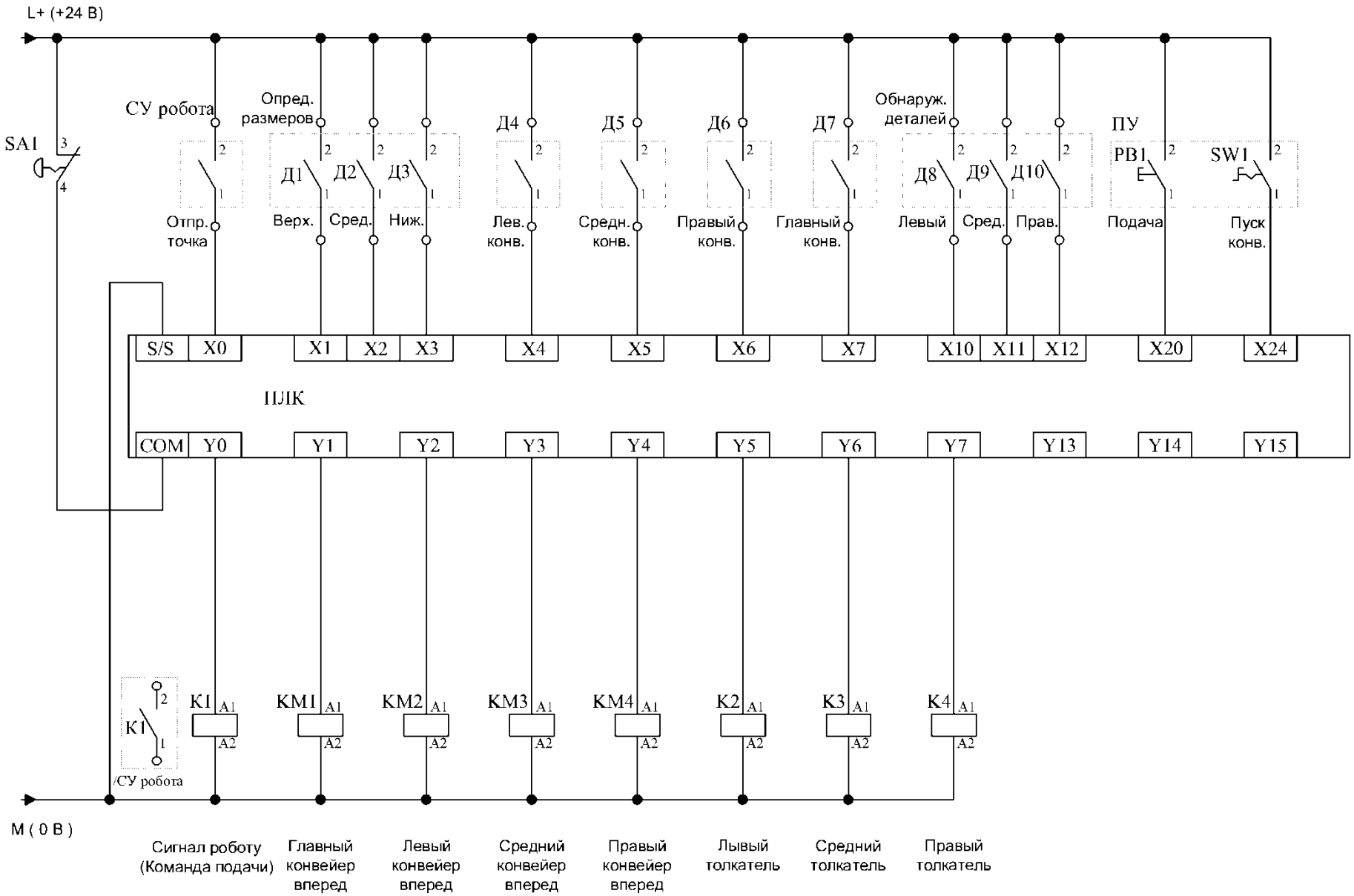



Рис. 10.4. Схема подключения входов-выходов ПЛК (упражнение F-3)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1 – автоматический выключатель для двигателя с тепловым реле
QF2…QF10 – автоматические выключатели
KM1 – контактор для управления двигателем (главный конвейер)
KM2 – контактор для управления двигателем (левый конвейер)
KM3 – контактор для управления двигателем (средний конвейер)
KM4 – контактор для управления двигателем (правый конвейер)
K1 – реле подачи управляющего сигнала роботу
K2, K3, K4 – реле для управления толкателями (левым, средним, правым соответственно)
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
Д1…Д10 – датчики положения

**Контрольные вопросы**

1. Синтаксис обращения к подпрограмме и возврата в главную программу.
2. Синтаксис обращения к программе обработки прерываний и в каких целях она используется?
3. Какую минимальную ширину импульса должны иметь сигналы прерывания и почему?
4. Какая из программ прерывания будет обрабатываться вначале в случае одновременного вызова нескольких программ прерывания?
5. Какая инструкция позволяет обрабатывать программу, время цикла которой превышает 200 мс?

# ЛАБОРАТОРНАЯ РАБОТА №11

## Отбраковка деталей

### Цель работы

Самостоятельно создать программное обеспечение для заданной в упражнении F-4 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

### Упражнение F-4 – Отбраковка деталей.

На рисунке 11.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

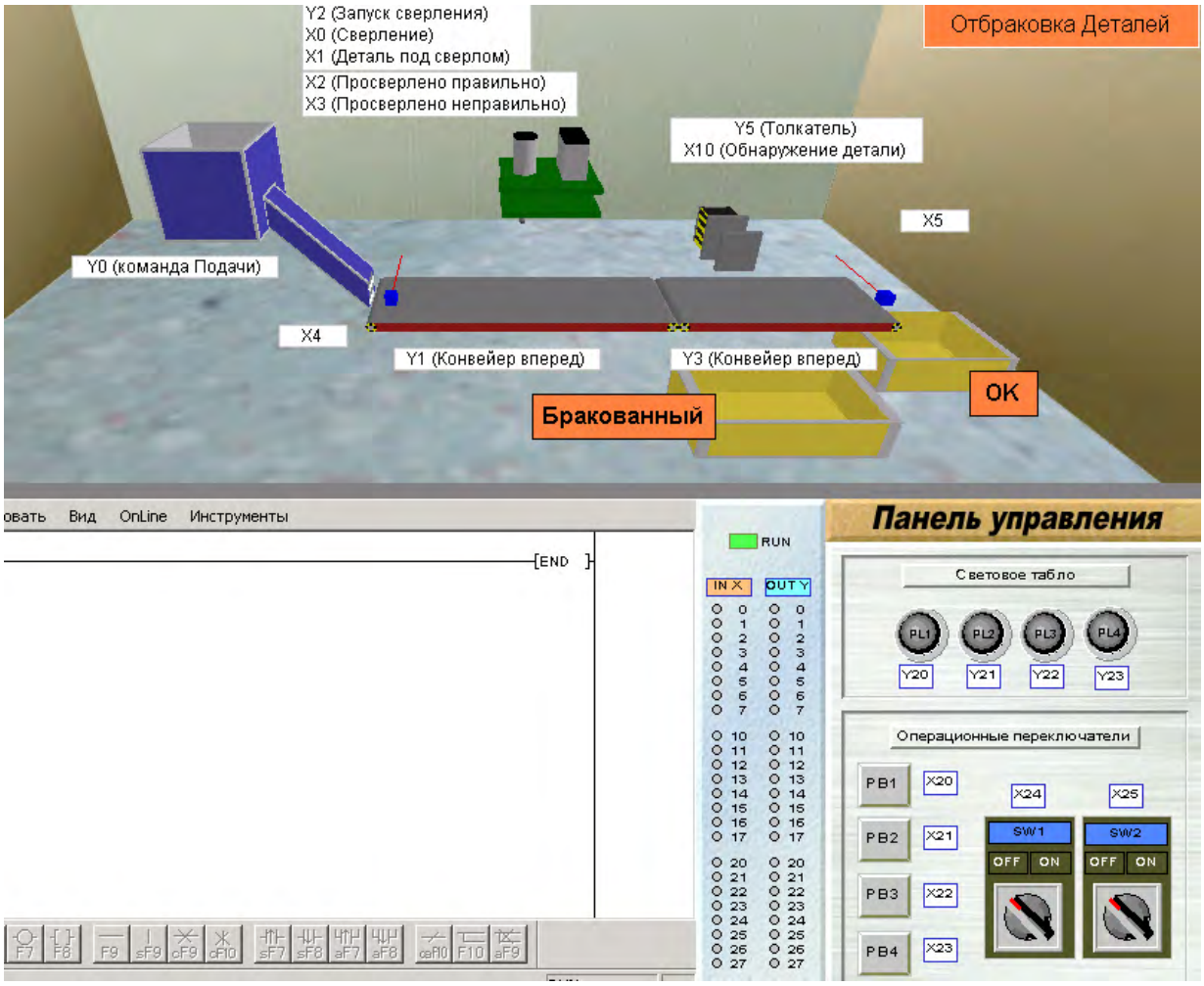



Рис. 11.1. Панель управления и виртуальное оборудование к упражнению F-4

Кнопка *PB1* (вход *X20* контроллера) задает для загрузочной воронки с выхода *Y0* управляющий сигнал подачи детали на конвейер. Тумблер *SW1* (вход *X24* контроллера) управляет пуском-остановом конвейеров с выходов *Y1* и *Y3* в положениях *ON/OFF* соответственно. Датчик *Деталь под сверлом (X1)*

фиксирует прохождение детали и останавливает конвейер для запуска процесса сверления *(Y2)*. После завершения цикла сверления считываются показания датчиков *X2 (Просверлено правильно)* и *X3 (Просверлено неправильно)*. В случае обнаружения бракованной детали встроенным в толкатель датчиком *Обнаружение детали (X10)* конвейер останавливается, и толкатель *(Y5)* сталкивает ее на поддон дефектных деталей. Хорошая деталь перемещается конвейером к правому поддону.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 11.2

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 11.3, 11.4.

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи разделения бракованных и хороших деталей в соответствии с сигналами датчиков с учетом следующих условий:

   Общее управление

   a) Когда на **Панели управления** нажата кнопка *PB1 (X20),* команда *Подачи (Y0)* для загрузочной воронки переходит в состояние *ON* и из загрузочной воронки подается деталь. Если кнопка *PB1(X20)* отпущена, команда *Подачи (Y0)* переходит в состояние *OFF*.
   b) Когда тумблер *SW1 (X24)* на **Панели управления** установлен в состояние *ON*, конвейеры движутся. Когда переключатель *SW1(X24)* установлен в состояние *OFF*, конвейеры останавливаются.

   Управление сверлением

   a) Когда установленный в сверлильной машине датчик *Деталь под сверлом (X1)* переходит в состояние *ON*, конвейеры останавливаются.
   b) Когда *Запуск сверления (Y2)* переходит в состояние *ON*, запускается процесс сверления. *Запуск сверления (Y2)* переключается в состояние *OFF*, когда датчик *Сверление (X0)* находится в состоянии *ON*.
   c) После того, как сверлильная машина отработала один полный цикл, установится в состояние *ON* сигнал датчика *Просверлено правильно (X2)* или датчика *Просверлено неправильно (X3)*. Сверлильная машина не может быть остановлена в середине цикла сверления. В этой имитации

одна из трех деталей должна иметь дефект. Если в детали просверлено несколько отверстий, то такая деталь также считается дефектной.

d) Когда дефектная деталь обнаружена встроенным в толкатель датчиком *Обнаружение детали (X10)*, конвейер останавливается и толкатель сталкивает ее на поддон дефектных деталей.

ЗАМЕЧАНИЕ

Когда команда приведения в действие толкателя установлена в состояние *ON*, он выдвигается полностью. В состоянии *OFF* толкатель полностью втягивается.

e) Хорошую деталь конвейер перемещает к поддону, расположенному с правой стороны.

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:

a) На **Панели управления** установите в состояние *ON* тумблер *SW1 (X24)*
Результат ► конвейер движется вправо.

b) На **Панели управления** нажмите кнопку *PB1 (X20)*
Результат ► деталь поступает из загрузочной воронки.

c) Сверление
Результат ►деталь останавливается под сверлом, и в ней просверливается отверстие.

d) Функционирование после сверления хорошей детали
Результат ► конвейер перемещает деталь к поддону, расположенному с правой стороны.

e) Функционирование после сверления дефектной детали
Результат ►деталь останавливается перед толкателем и выталкивается на поддон для бракованных деталей.

f) Повторение функционирования
Результат ►Если нажата кнопка *PB1 (X20),* то функционирование повторяется, начиная с пункта b).
Для того чтобы иметь возможность повторить функционирование, следует инициализировать экран посредством щелчка на дистанционном управлении по кнопке *Reset*.

3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-4

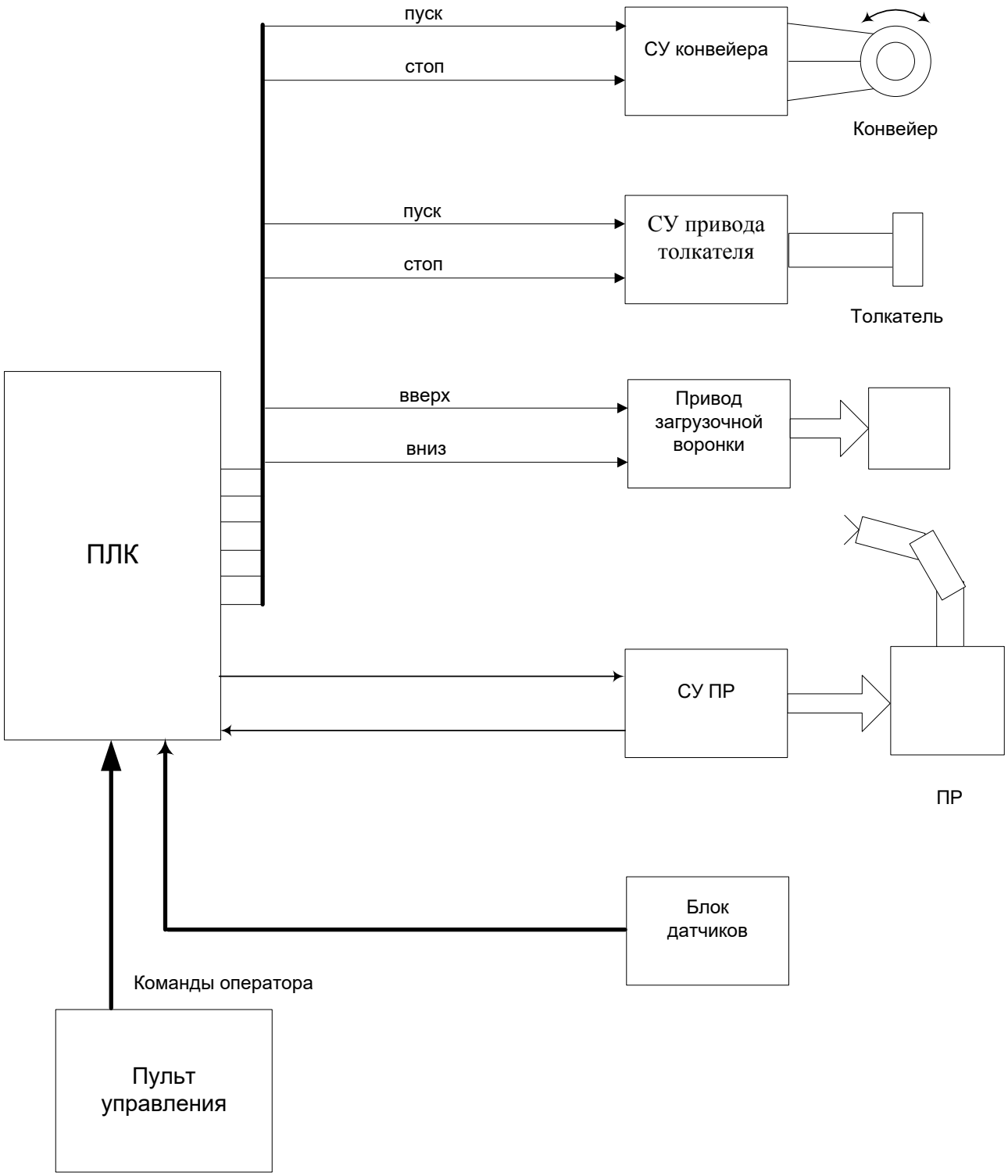



Рис. 11.2. Структурная схема СУ к упражнению F-4

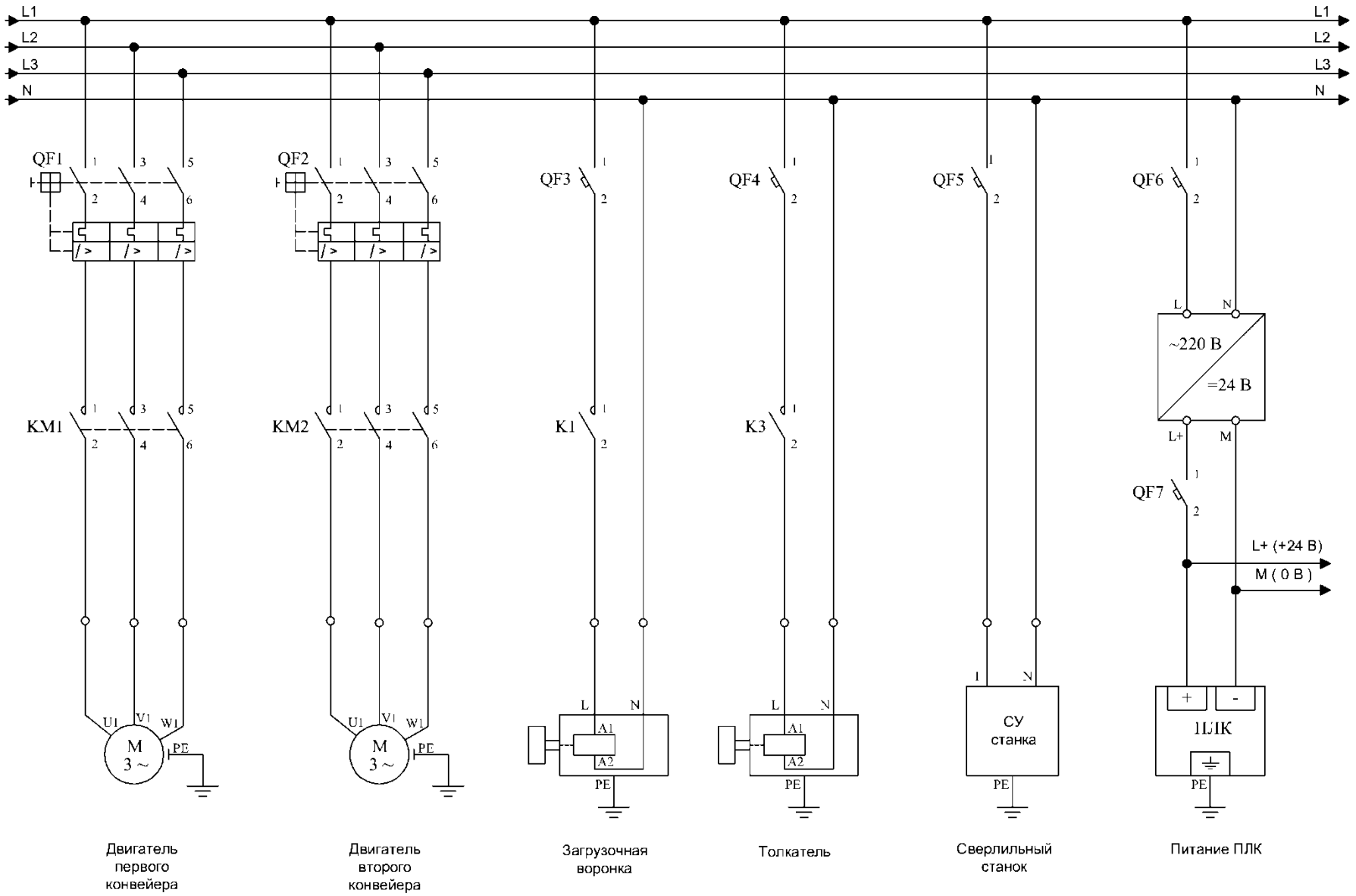



Рис. 11.3. Схема подключения оборудования (упражнение F-4)

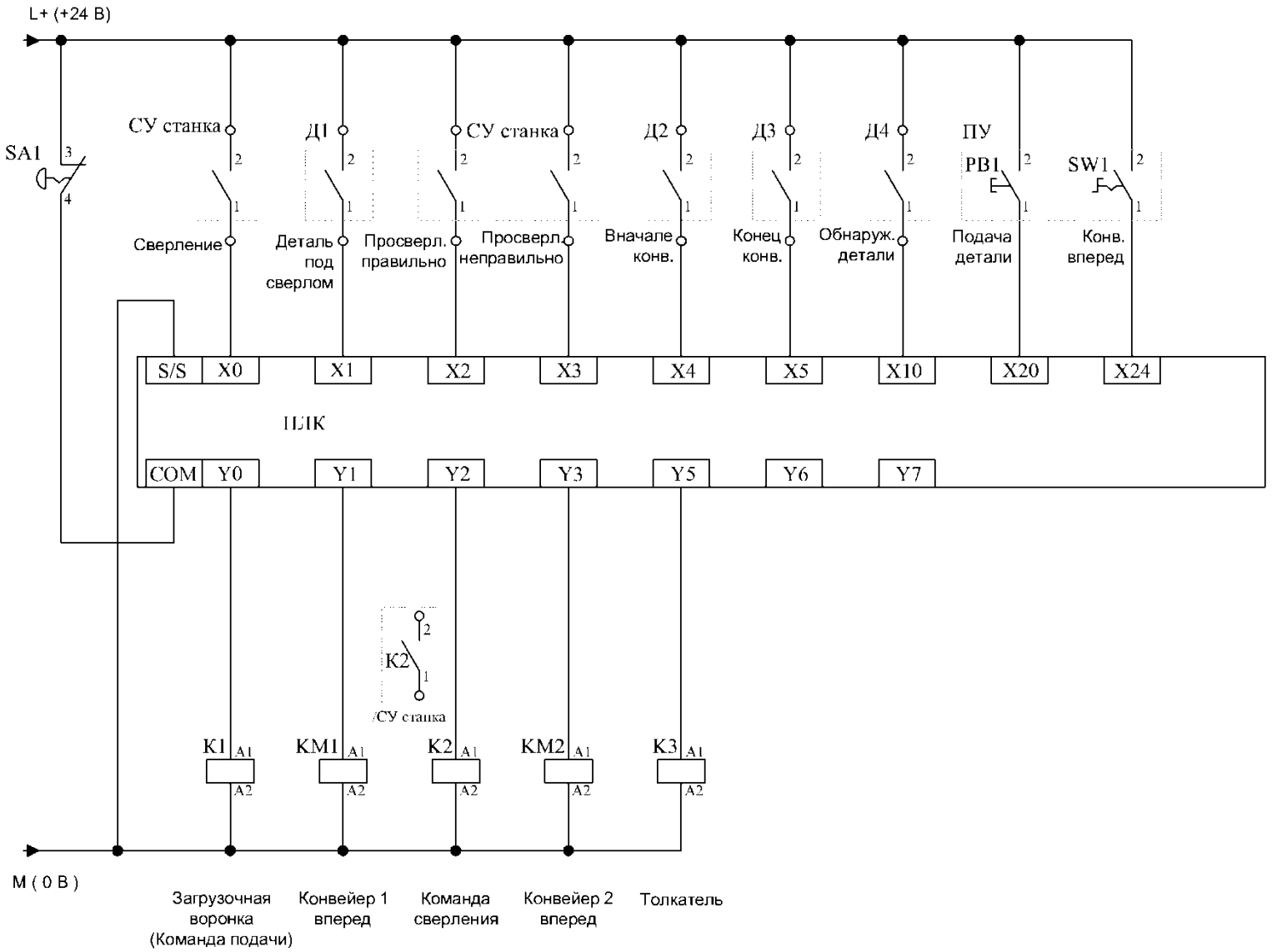



Рис. 11.4. Схема подключения ПЛК входов-выходов ПЛК (упражнение F-4)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF2 – автоматический выключатель для двигателя с тепловым реле
QF3…QF7 – автоматические выключатели
KM1 – контактор для управления двигателем (конвейер 1 вперед)
KM2 – контактор для управления двигателем (конвейер 2 вперед)
K1 – реле для подачи управления загрузочной воронкой
K2 – реле для подачи управляющего сигнала СУ сверлильного станка
K3 – реле для управления толкателем
Д1…Д4 – датчики положения
PB1 – кнопка на панели управления
SW1 – тумблер на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Какая команда инвертирует результат обработки предыдущей команды?
2. С помощью какой команды можно организовать пошаговую проверку программы?
3. Каким образом осуществляется маркировка точки в программе?
4. Таймеры каких порядковых номеров могут быть использованы в подпрограмме?
5. Синтаксис задания цикла в программе.

# ЛАБОРАТОРНАЯ РАБОТА №12

## Управление движением конвейера согласно определенному размеру детали

### Цель работы

Самостоятельно создать программное обеспечение для заданной в упражнении F-5 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

### Упражнение F-5 – Управление движением конвейера.

На рисунке 12.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

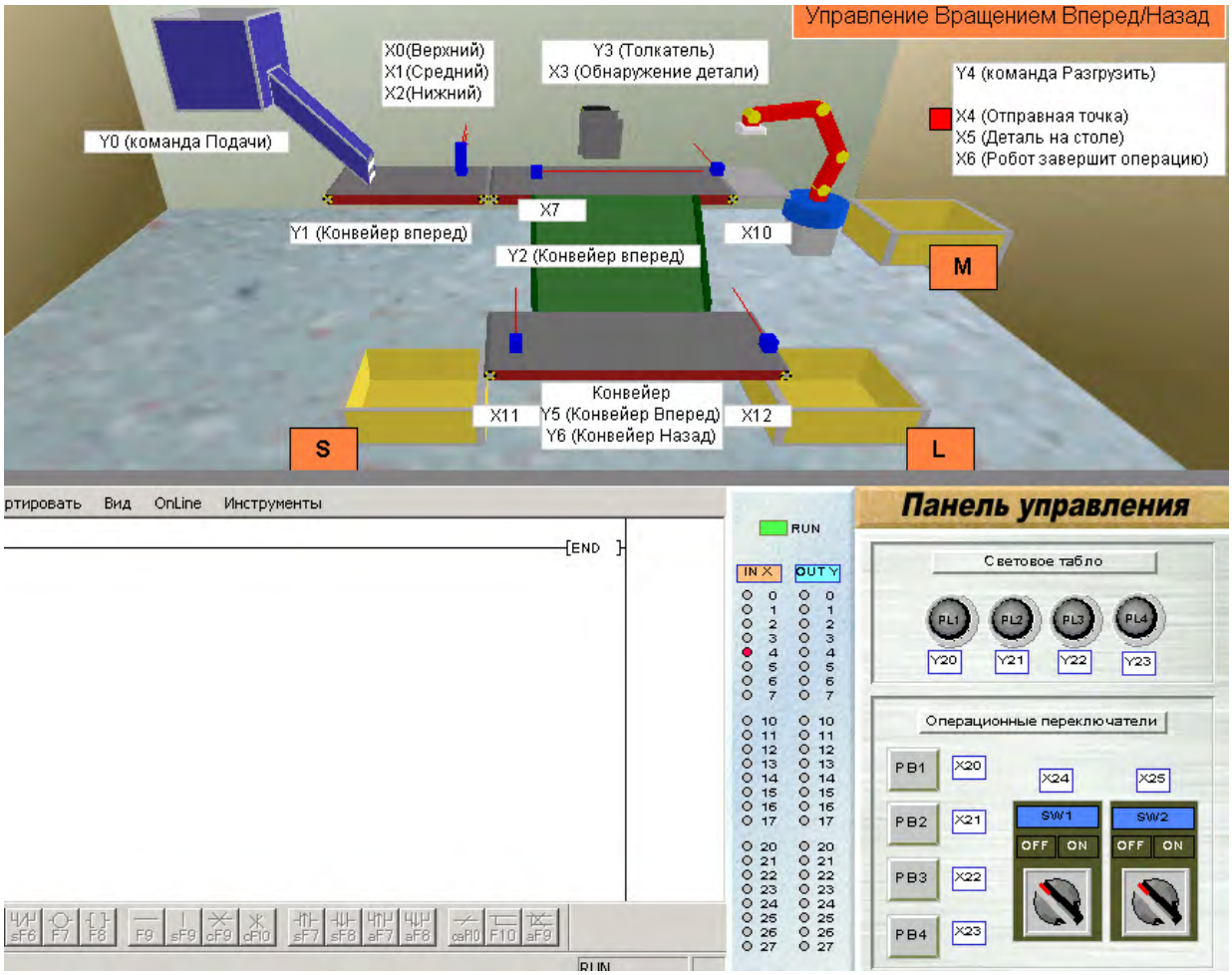



Рис. 12.1. Панель управления и виртуальное оборудование к упражнению F-5

Кнопка *PB1* (вход *X20* контроллера) задает для загрузочной воронки с выхода *Y0* управляющий сигнал подачи детали на конвейер. Тумблер *SW1* (вход *X24* контроллера) управляет пуском-остановом конвейеров с выходов *Y1, Y2* и *Y5* в положениях *ON/OFF* соответственно. Датчики *X0, X1, X2* фиксируют прохождение деталей большой, средней и малой величины соответственно.

После сортировки по размерам деталей по сигналам от датчиков осуществляется их перенос к определенным поддонам. Детали большой и малой величины сталкиваются толкателем *Y5* на нижний конвейер и переносятся к правому поддону L для больших или к левому поддону S для малых деталей. Детали средней величины переносятся в поддон M роботом.

Тумблер *SW2* (вход *X25* контроллера) в положении *ON* обеспечивает автоматическую подачу деталей из загрузочной воронки в случае, когда робот начинает переносить в поддон M деталь среднего размера, либо когда большая или малая деталь помещена в поддон.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 12.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными и однофазными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейеров. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 12.3, 12.4.

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи управления движением конвейера согласно определенному размеру детали с учетом следующих условий:
   a) Когда на **Панели управления** нажата кнопка *PB1 (X20),* команда *Подачи (Y0)* для загрузочной воронки переходит состояние *ON* и осуществляется подача детали. Когда кнопка *PB1(X20)* отпущена, команда *Подачи (Y0)* переходит в состояние *OFF*.
   b) Когда тумблер *SW1(X24)* на **Панели управления** установлен в положение *ON*, конвейеры движутся. Когда тумблер *SW1(X24)* установлен в положение *OFF*, конвейеры останавливаются.
   c) Большие, средние и мелкие детали на конвейерах сортируются по сигналам, поступающим от датчиков *Верхний (X0), Средний (X1)* и *Нижний (X2)* и переносятся к установленным для каждого размера поддонам.
      Большая деталь: сталкивается на нижний конвейер и переносится к правому поддону L.
      Средняя деталь: переносится к поддону M роботом.

Малая деталь: сталкивается на нижний конвейер и переносится к левому поддону S.

d) Когда датчик *Обнаружение детали (X3)* переходит в состояние *ON*, конвейер останавливается и большая или малая деталь сталкивается на нижний конвейер.

ЗАМЕЧАНИЕ

Когда команда приведения в действие толкателя установлена в *ON*, он выдвигается полностью. В состоянии *OFF* толкатель полностью втягивается.

e) Когда состояние установленного в роботе датчика *Деталь на столе (X5)* перейдет в состояние *ON*, команда *Разгрузить (Y4)* устанавливается в состояние *ON* и робот переносит деталь к поддону M. Когда *Функционирование робота окончено (X6)* перейдет в состояние *ON* (состояние *ON* соответствует размещению детали на поддоне), команда *Разгрузить (Y4)* устанавливается в состояние *OFF*.

f) Если на **Панели управления** тумблер *SW2(X25)* установлен в положение *ON*, осуществляется автоматическая подача детали в момент, когда:

- робот начинает переносить деталь среднего размера;
- малая или большая деталь помещена на поддон.

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:

a) На **Панели управления** установите в положение *ON* тумблер *SW1(X24)*
Результат ► конвейер движется вправо.

b) На **Панели управления** нажмите кнопку *PB1 (X20)*
Результат ► деталь поступает из загрузочной воронки.

c) Сортировка деталей по размеру
Результат ► Большая деталь: сталкивается на нижний конвейер и переносится к правому поддону L.
Средняя деталь: переносится к поддону M роботом.
Малая деталь: сталкивается на нижний конвейер и переносится к левому поддону S.

d) Функционирование во время установки тумблера *SW2 (X25)* на **Панели управления** в положение *ON*
Результат ► очередная деталь поступает из загрузочной воронки, когда робот начинает перемещать среднюю деталь или когда малая или большая деталь окажется на поддоне.

3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-5

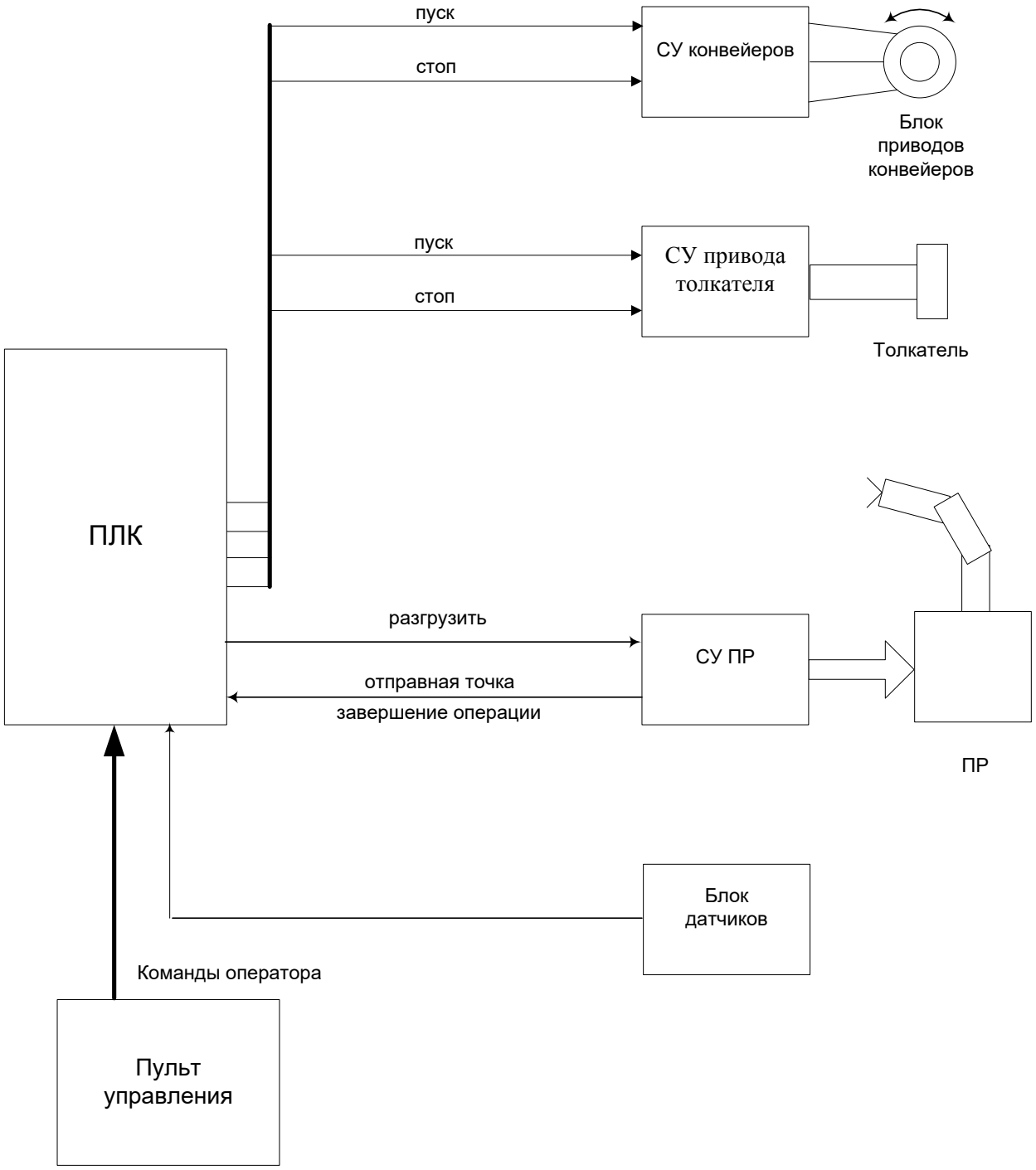



Рис. 12.2. Структурная схема СУ к упражнению F-5

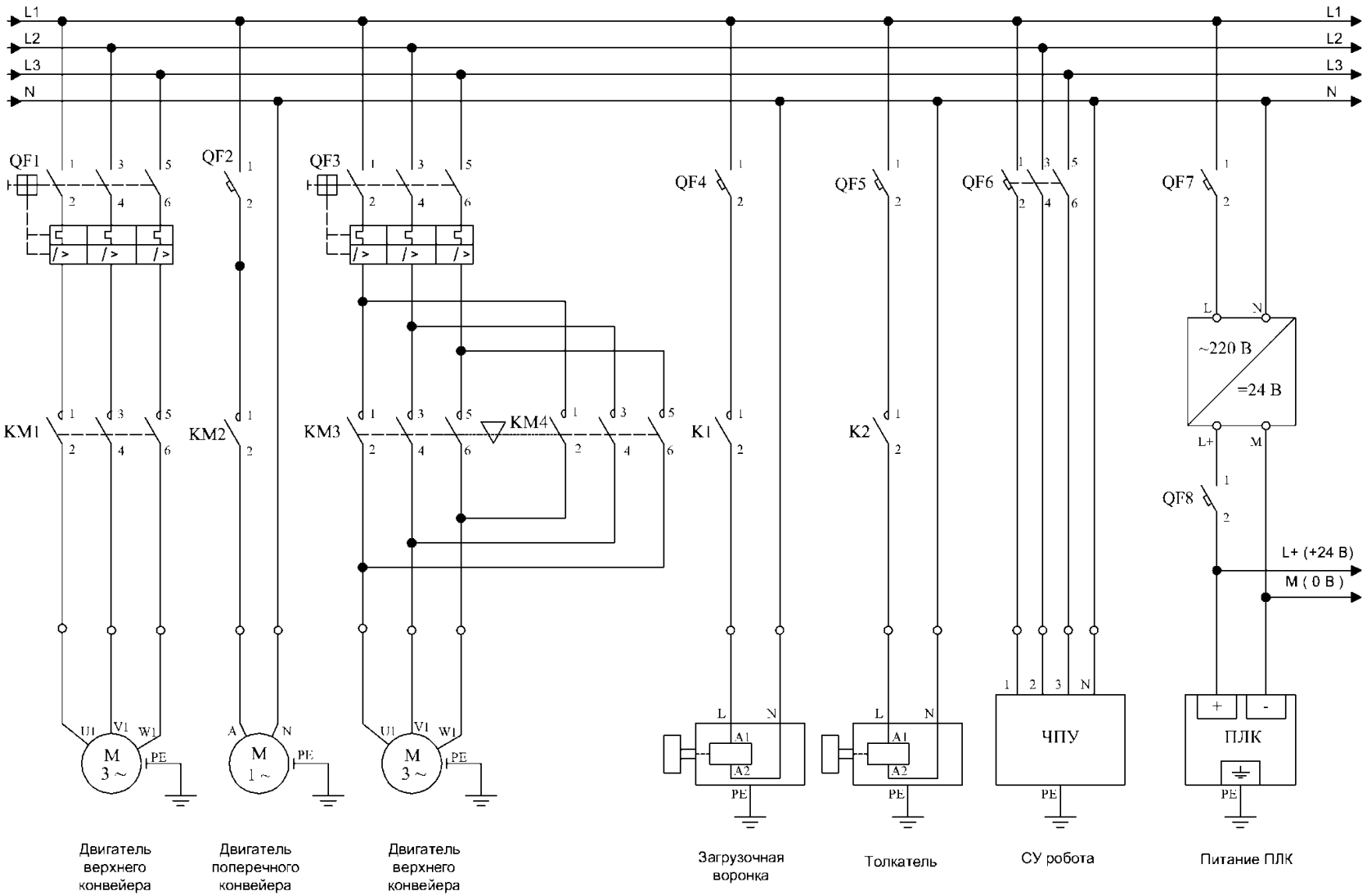



Рис. 12.3. Схема подключения оборудования (упражнение F-5)

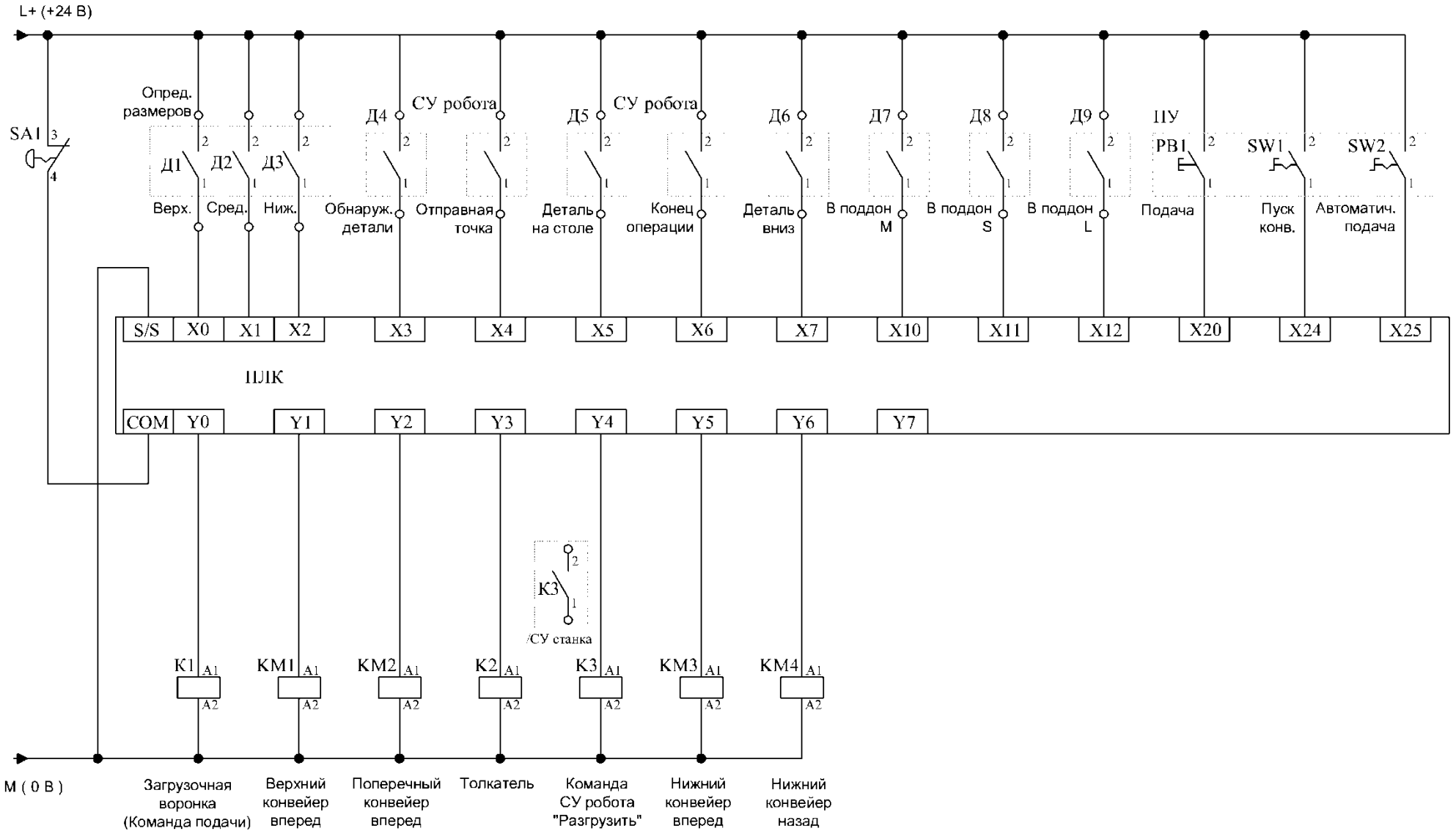



Рис. 12.4. Схема подключения входов-выходов ПЛК (упражнение F-5)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1, QF3 – автоматический выключатель для двигателя с тепловым реле
QF2, QF4…QF8 – автоматические выключатели
KM1 – контактор для управления двигателем (верхний конвейер вперед)
KM2 – контактор для управления двигателем (поперечный конвейер вперед)
KM3 – контактор для управления двигателем (нижний конвейер вперед)
KM4 – контактор для управления двигателем (нижний конвейер назад)
K1 – реле для подачи управления загрузочной воронкой
K2 – реле для управления толкателем
K3 – реле для подачи управляющего сигнала СУ робота («Разгрузить»)
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)
Д1…Д3 – датчики определения размера детали
Д4…Д9 – датчики положения
PB1 – кнопка на пульте управления («Подача детали»)
SW1, SW2 – тумблеры на панели управления

**Контрольные вопросы**

1. Назовите функции регистров отображения входов и выходов.
2. Инструкция обновления регистров отображения.
3. Инструкции использования высокоскоростного счетчика.
4. Синтаксис выдачи определенного числа импульсов с жестко заданной частотой и соотношением ширины импульса 50:50.
5. Синтаксис выдачи импульсов с жестко заданной шириной импульса и продолжительностью периода.
6. Синтаксис выдачи определенного числа импульсов с заданной частотой.

# ЛАБОРАТОРНАЯ РАБОТА №13

## Управление подъемным приспособлением

### Цель работы

Самостоятельно создать программное обеспечение для заданной в упражнении F-6 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

### Упражнение F-6 – Управление подъемным приспособлением.

На рисунке 13.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

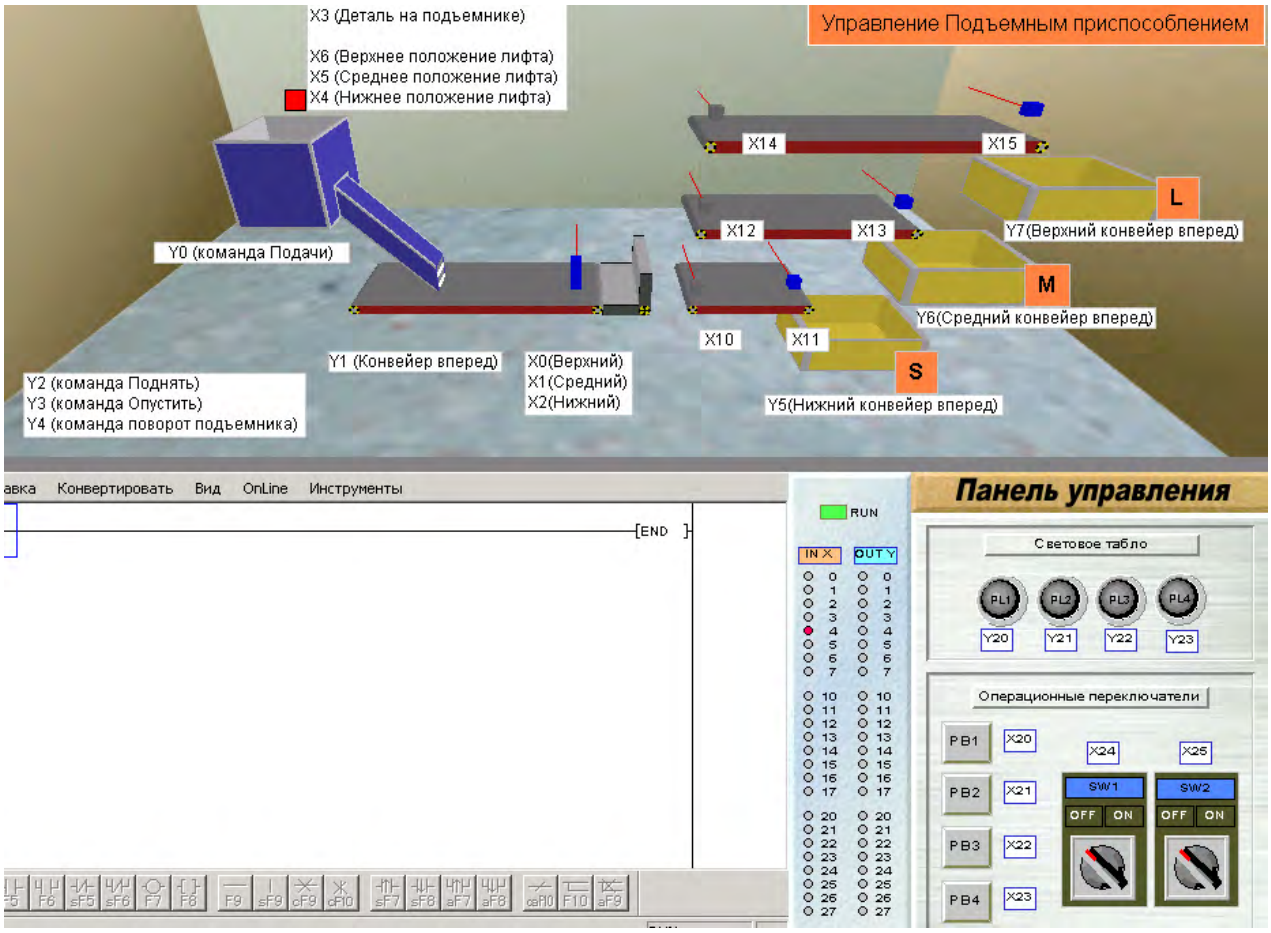



Рис. 13.1. Панель управления и виртуальное оборудование к упражнению F-6

Кнопка *PB1* (вход *X20* контроллера) задает для бункера с выхода *Y0* контроллера управляющий сигнал подачи детали на конвейер. Тумблер *SW1* (вход *X24* контроллера) управляет пуском-остановом конвейера с выхода *Y1* в положениях *ON/OFF* соответственно. Датчики *Верхний (X0), Средний (X1)* и *Нижний (X2)* фиксируют прохождение деталей большой, средней и малой величины соответственно. Датчик *Деталь на подъемнике (X3)* фиксирует

нахождение детали на подъемнике и в соответствии с показаниями датчиков *X0, X1, X2* переносит деталь на верхний, средний или нижний конвейер. После того, как один из датчиков *X10, X12* или *X14* с левого конца конвейера обнаружит деталь, соответствующий конвейер переходит в состояние *ON* и переносит деталь к поддону с правой стороны. Конвейер останавливается спустя 3 секунды после того, как деталь пройдет мимо датчика *X11, X13* или *X15* в правом конце конвейера. *Подъемом (Y2)* и *опусканием (Y3)* подъемника управляют сигналы датчиков *Нижнее (X4), Среднее (X5)* и *Верхнее положение подъемника (X6)*. Когда деталь перенесена с подъемника на конвейер, переходит в состояние *ON* команда *Вращения подъемника (Y4)* и он возвращается в начальное положение.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 13.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках13.3, 13.4.

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи управления движением подъемника с учетом следующих условий:

   Общее управление

   a) Когда на **Панели управления** нажата кнопка *PB1(X20)* команда *Подачи (Y0)* для загрузочной воронки переходит состояние *ON* и осуществляется подача детали. Когда кнопка *PB1(X20)* отпущена, команда *Подачи (Y0)* переходит в состояние *OFF*.

   b) Когда тумблер *SW1 (X24)* на **Панели управления** установлен в положение *ON*, конвейер, на который поступают детали из загрузочной воронки, движется. Когда тумблер *SW1 (X24)* установлен в положение *OFF*, конвейер останавливается.

   c) После того, как один из датчиков *X10, X12* или *X14* с левого конца конвейера обнаружит деталь, соответствующий конвейер переходит в состояние *ON* и переносит деталь к поддону с правой стороны. Конвейер останавливается спустя 3 секунды после того, как деталь пройдет мимо датчика *X11, X13* или *X15* в правом конце конвейера.

   d) Большие, средние и мелкие детали на конвейере сортируются по сигналам, поступающим от датчиков *Верхний (X0), Средний (X1)* и *Нижний (X2)*

   Управление подъемником

   a) Когда встроенный в подъемник датчик *Деталь на подъемнике (X3)* переходит в состояние *ON*, осуществляется перенос детали согласно ее размеру к одному из следующих конвейеров:

      Большая деталь: к верхнему конвейеру.

      Средняя деталь: к среднему конвейеру.

      Малая деталь: к нижнему конвейеру.

   *b)* Команды *Подъемник вверх (Y2)* и *Подъемник вниз (Y3)* подаются согласно положению подъемника, определяемому датчиками *Верхний (X6), Средний (X5)* и *Нижний (X4)*.

   c) Когда деталь перенесена с подъемника на конвейер, переходит в состояние ON команда *Вращения подъемника (Y4)*.

   d) После того, как деталь перенесена, подъемник должен возвратиться в начальное положение и находиться в ожидании.

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:

   a) На **Панели управления** установите в положение *ON* тумблер *SW1 (X24)*
   Результат ► конвейер движется вправо.

   b) На **Панели управления** нажмите кнопку PB1 (X20)
   Результат ► из загрузочной воронки подается деталь.

   c) Функционирование подъемника

Результат ► Когда деталь большого или среднего размера установлена на подъемник, то он поднимается до верхнего или среднего конвейера и останавливается. Когда деталь малой величины установлена на подъемник, он остается в начальном положении.

d) Перенос детали
   Результат ► Подъемник разворачивается и устанавливает деталь на конвейер, который переносит ее вправо к поддону.
e) Возвращение подъемника
   Результат ► подъемник без детали возвращается в начальное положение и находится в ожидании.
f) Повторение функционирования
   Результат ► Когда нажата кнопка *PB1 (X20),* функционирование повторяется, начиная с шага b).

3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-6

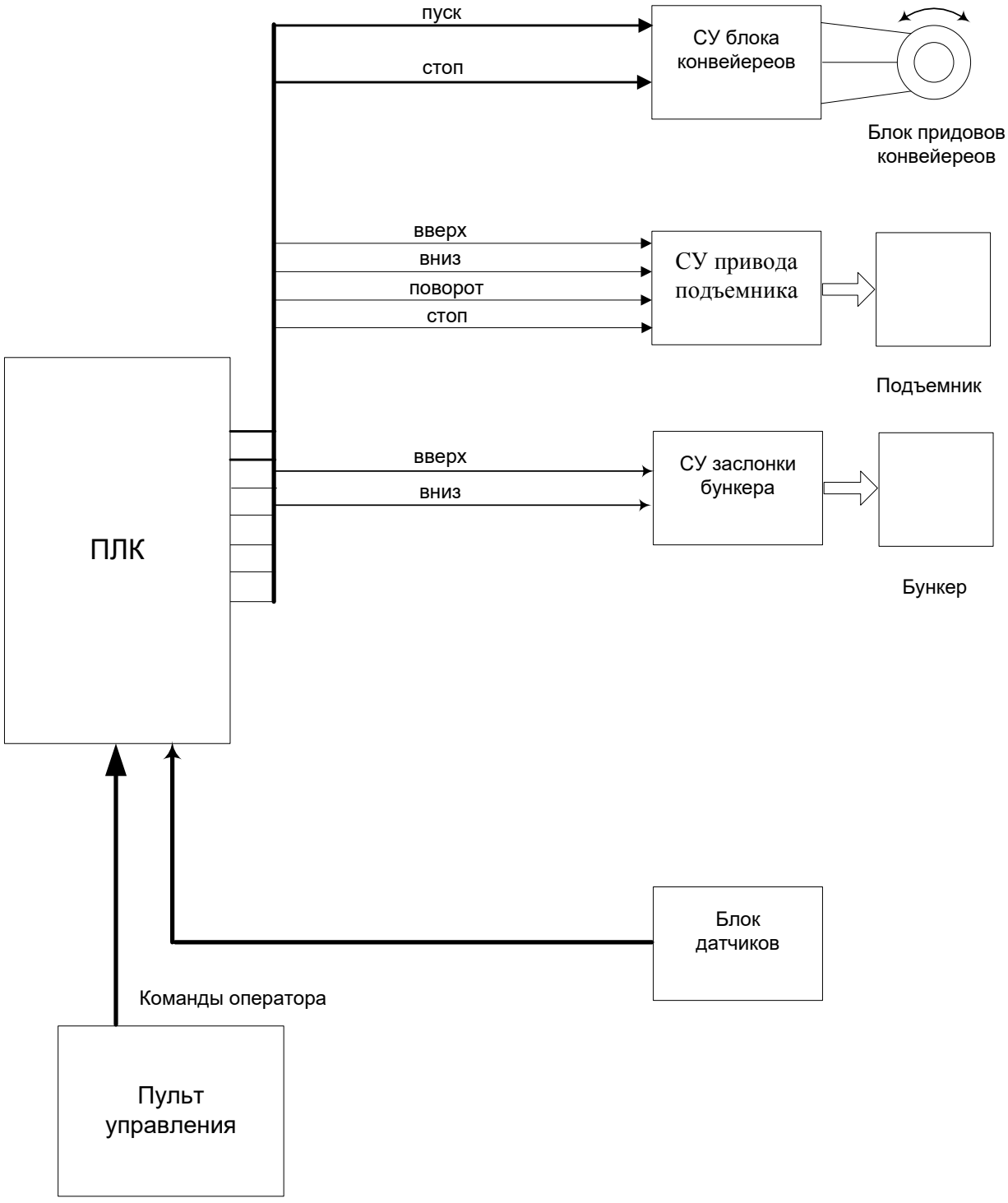



Рис. 13.2. Структурная схема СУ к упражнению F-6

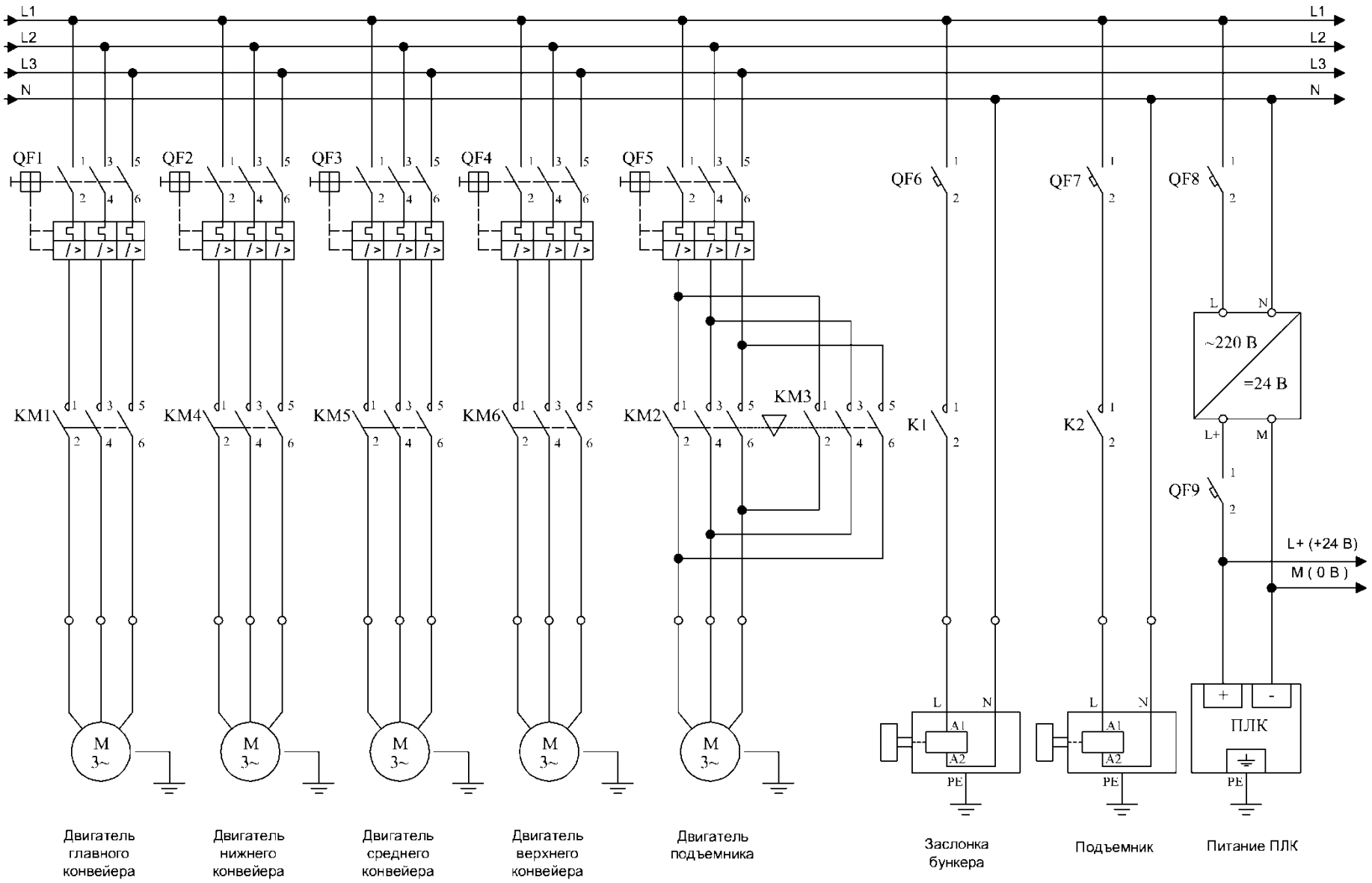



Рис. 13.3. Схема подключения оборудования (упражнение F-6)

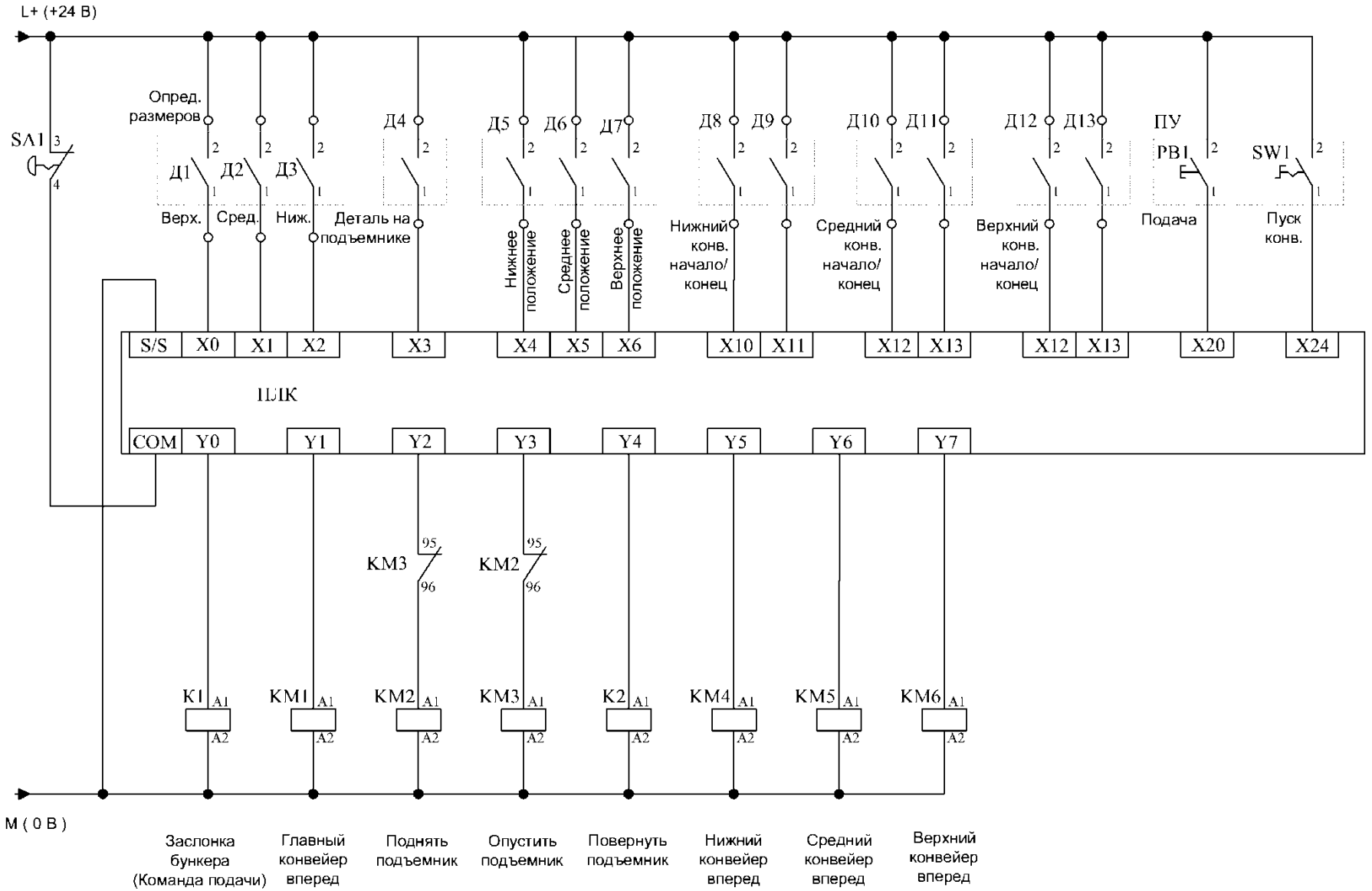



Рис. 13.4. Схема подключения входов-выходов ПЛК (упражнение F-6)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1…QF5 – автоматические выключатели для двигателя с тепловым реле
QF6…QF9 – автоматические выключатели
KM1 – контактор для управления двигателем (главный конвейер вперед)
KM2 – контактор для управления двигателем (поднять подъемник)
KM3 – контактор для управления двигателем (опустить подъемник)
KM4 – контактор для управления двигателем (нижний конвейер вперед)
KM5 – контактор для управления двигателем (средний конвейер вперед)
KM6 – контактор для управления двигателем (верхний конвейер вперед)
K1 – реле для подачи управляющего сигнала заслонки бункера
K2 – реле для управления поворотом подъемника
Д1…Д3 – датчики определения размера детали
Д4…Д13 – датчики положения
PB1 – кнопка на пульте управления («Подача детали»)
SW1, SW2 – тумблеры на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Инструкции для программирования шагового управления.
2. Какие виды STL-разветвлений Вы знаете?
3. Команды сложения и вычитания двух числовых данных.
4. Команды умножения двух числовых данных. Команды умножения 16-ти битных и 32-х битных данных.
5. Команды деления двух числовых данных. Команды деления 16-ти битных и 32-х битных данных.
6. Синтаксис команд приращения и уменьшения.
7. Назначение и функциональные возможности SCADA-систем.

## ЛАБОРАТОРНАЯ РАБОТА №14

### Линия сортировки и распределения

**Цель работы**

Самостоятельно разработать программное обеспечение для заданной в упражнении F-7 СУ, используя изученные ранее инструкции, применяемые при программировании контроллеров.

**Упражнение F-7 – линия сортировки и распределения.**

На рисунке 14.1 показана **Панель управления** с индикаторами состояния входов-выходов ПЛК, кнопками и тумблерами, с помощью которых задаются сигналы управления виртуальным оборудованием.

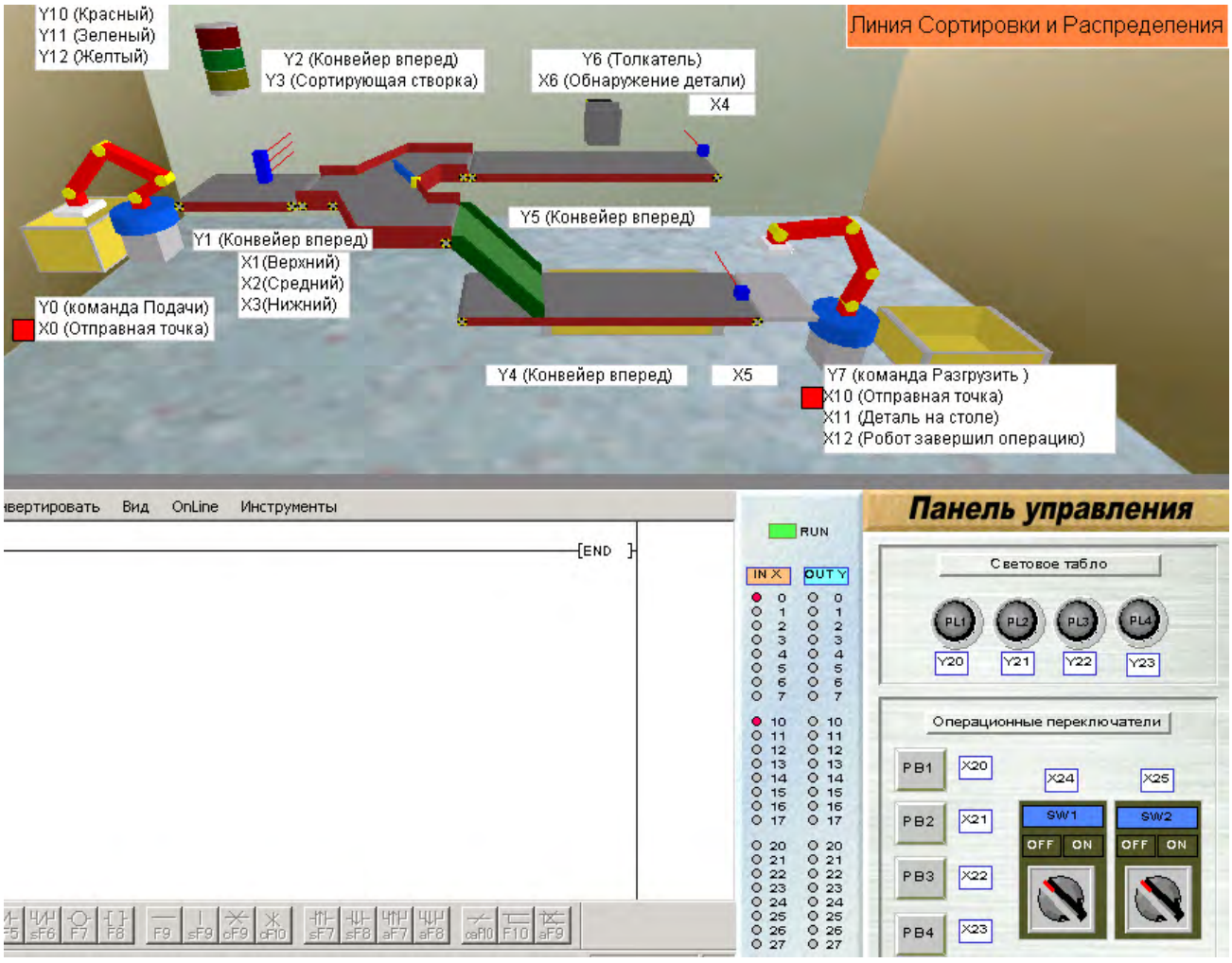



Рис. 14.1. Панель управления и виртуальное оборудование к упражнению F-7

Кнопка *PB1* (вход *X20* контроллера) задает для робота с выхода *Y0* управляющий сигнал подачи детали на конвейер. Тумблер *SW1* (вход *X24* контроллера) управляет пуском-остановом конвейеров с выходов *Y1, Y2, Y4* и *Y5* в положениях *ON/OFF* соответственно. Датчики *Верхний (X1), Средний (X2)* и

*Нижний (X3)* фиксируют прохождение деталей большой, средней и малой величины соответственно. Детали большой и малой величины посредством *Сортирующей створки (Y3)* переносятся к тыловому конвейеру, после чего большая деталь падает с правой стороны от конвейера, а малая сталкивается *толкателем (Y6)* в поддон. Детали средней величины переносятся к переднему конвейеру и затем помещаются роботом в поддон.

Тумблер *SW2* (вход *X25* контроллера) в положении *ON* обеспечивает автоматическую подачу деталей в случае, когда робот-разгрузчик начинает переносить деталь среднего размера в поддон, либо когда малая деталь помещена в поддон или большая упала с правой стороны конвейера.

Для реализации поставленной задачи управления технологическим оборудованием предлагается система управления, структурная схема которой показана на рисунке 14.2.

Основным управляющим элементом системы является ПЛК, который по сигналам блока датчиков реализует управление технологическим оборудованием. Выходные сигналы датчиков поступают на входы контроллера. Обработка сигналов датчиков осуществляется программно. Конвейеры приводятся в движение трехфазными асинхронными двигателями с релейно-контактной СУ. С выходов контроллер инициирует запуск и остановку конвейера. Запуск программы промышленного робота осуществляется с выхода контроллера сигналом, поступающим на вход системы управления роботом. Дальнейшая работа ПР осуществляется по составленной для него программе. По окончании цикла загрузки и возврату в исходную позицию с СУ ПР выдается сигнал “Конец цикла”, поступающий на вход ПЛК.

Схемы подключения оборудования и входов-выходов ПЛК приведены на рисунках 14.3, 14.4.

**Задание**

1. Разработать управляющую программу для реализации поставленной задачи сортировки деталей в точно установленное место в соответствии с их размерами с учетом следующих условий:
   a) Когда на **Панели управления** нажата кнопка *PB1(X20)*, команда *Подачи (Y0)* для робота переходит в состояние *ON* и робот подает деталь. Команда *Подачи (Y0)* переходит в состояние *OFF*, когда робот закончил перемещать деталь и возвратился в отправную точку.
   b) Когда на **Панели управления** тумблер *SW1 (X24)* переведен в положение *ON*, конвейеры движутся. Когда тумблер *SW1 (X24)* установлен в положение *OFF*, конвейеры останавливаются.
   c) Большие, средние и мелкие детали на конвейерах сортируются по сигналам, поступающим от датчиков *Верхний (X1)*, *Средний (X2)* и *Нижний (X3)* и переносятся к установленным для каждого размера детали конвейерам и поддонам.

      Большая деталь: Переносится к тыловому конвейеру, когда *Сортирующая створка (Y3)* конвейера с отводами

установлена в состояние *ON* и далее, перемещаясь по конвейеру, падает с правой стороны.

Средняя деталь: Переносится к переднему конвейеру, когда *Сортирующая створка (Y3)* конвейера с отводами установлена в состояние *OFF* и далее роботом-разгрузчиком переносится в поддон.

Малая деталь: Переносится к тыловому конвейеру, когда *Сортирующая створка (Y3)* конвейера с отводами установлена в состояние *ON*. Когда датчик *Обнаружения детали (X6)* в конвейере с отводами фиксирует прохождение детали, конвейер останавливается и деталь сталкивается в поддон.

d) Когда встроенный в робот-разгрузчик датчик *Деталь на столе (X11)* переходит в состояние ON, команда *Разгрузить (Y7)* устанавливается в состояние *ON* и робот-разгрузчик переносит деталь в поддон. Когда деталь помещена в поддон, команда *Разгрузить (Y7)* переходит в состояние *OFF.*

e) Если тумблер *SW2 (X25)* на **Панели управления** переключен в положение *ON*, очередная деталь автоматически подается роботом в момент когда:
- Робот-разгрузчик начинает переносить деталь среднего размера;
- Малая деталь помещена в поддон или большая упала с правой стороны конвейера.

f) Проблесковые огни зажигаются следующим образом:
- Красный горит во время загрузки детали роботом;
- Зеленый горит во время перемещения конвейера;
- Желтый горит во время остановки конвейера.
-

2. Выполнить проверку соответствия программы заданным условиям посредством 3D- **графической имитации**. Подтверждение соответствия программы провести по следующим пунктам:

a) На **Панели управления** установите в положение *ON* тумблер *SW1 (X24)*
Результат ► конвейер перемещается вправо.

b) На **Панели управления** нажните кнопку *PB1 (X20)*
Результат ►робот выполняет подачу детали.

c) Сортировка деталей по размеру
Результат ► Большая деталь: Переносится к тыловому конвейеру и падает с правой стороны.
Средняя деталь: Переносится к переднему конвейеру и помещается роботом-разгрузчиком в поддон.
Малая деталь: Переносится к тыловому конвейеру и сталкивается толкателем в нижний поддон.

d) Функционирование во время установки тумблера *SW2 (X25)* на **Панели управления** в положение *ON*

Результат ► очередная деталь автоматически подается роботом в момент, когда:

- Робот-разгрузчик начинает переносить деталь среднего размера;
- Малая деталь помещена в поддон или большая упала с правой стороны конвейера.

3. Разработать схему алгоритма управляющей программы.

**Структурная схема системы управления** к упражнению F-7

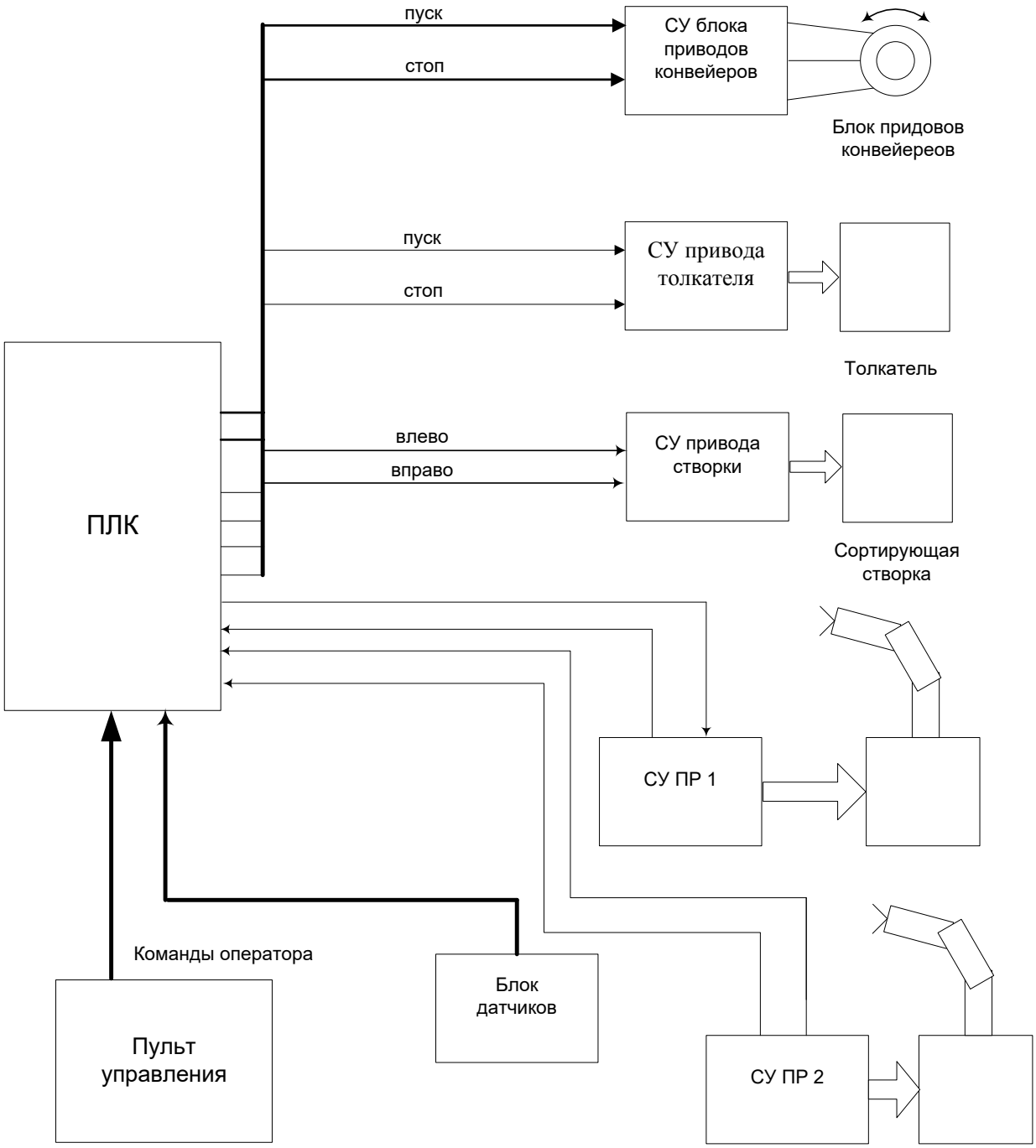



Рис. 14.2. Структурная схема СУ к упражнению F-7

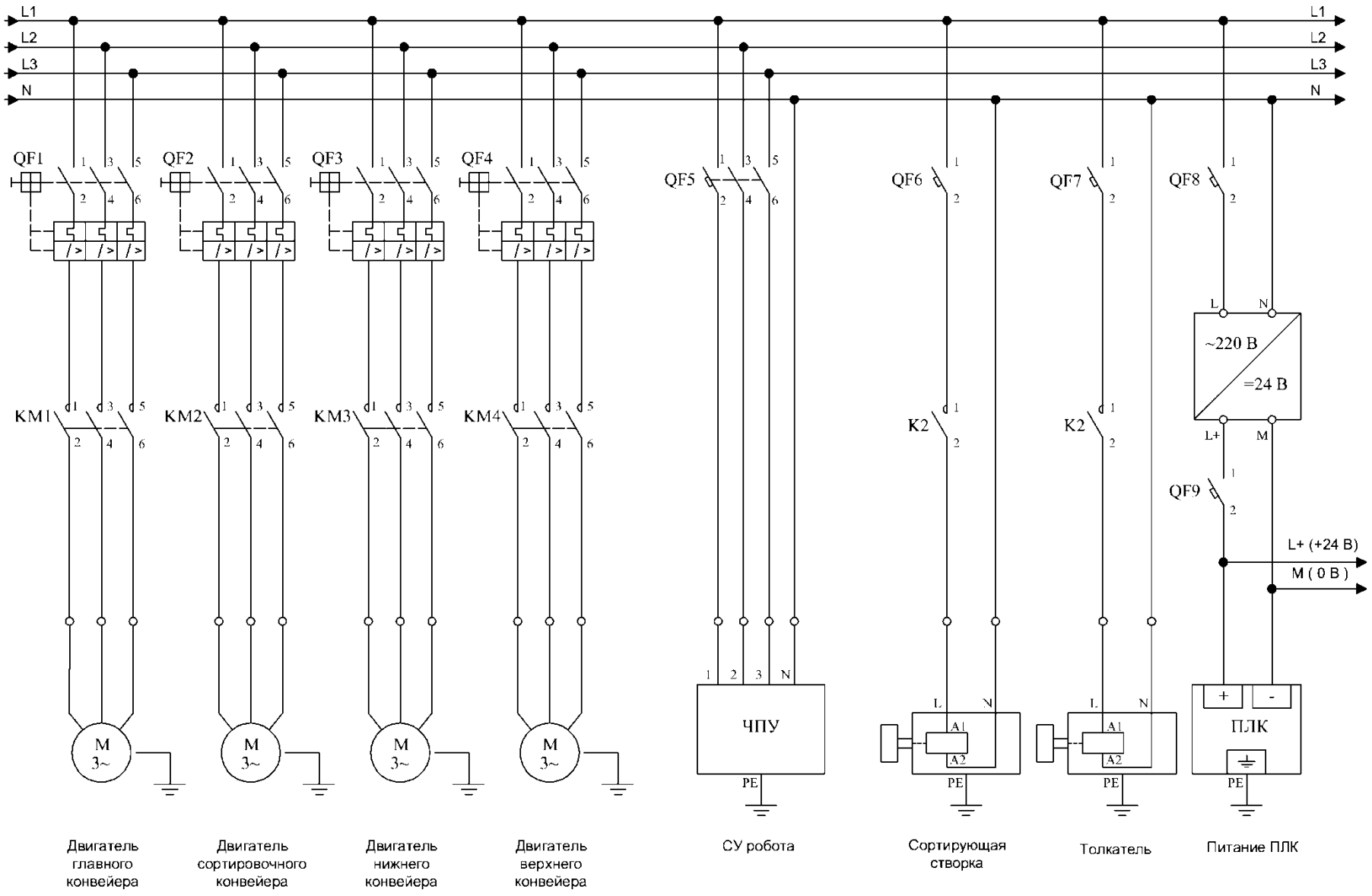



Рис. 14.3. Схема подключения оборудования (упражнение F-7)

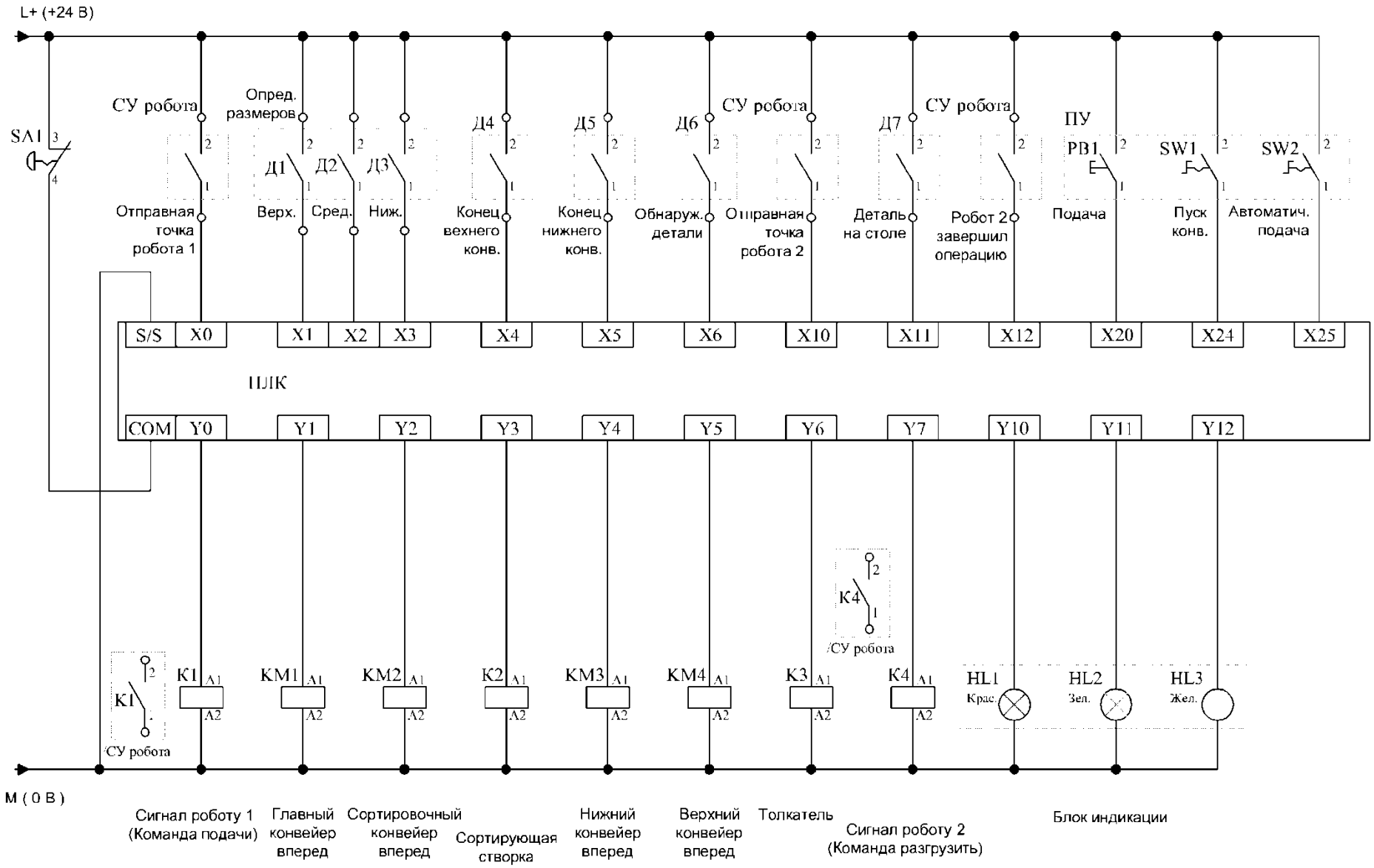



Рис. 14.4. Схема подключения входов-выходов ПЛК (упражнение F-7)

L1, L2, L3, N – трехфазный источник питания (380В, 50Гц)
GV1 – источник питания постоянного тока +24В
PE – провод заземления
QF1…QF4 – автоматические выключатели для двигателя с тепловым реле
QF5…QF9 – автоматические выключатели
KM1 – контактор для управления двигателем (главный конвейер вперед)
KM2 – контактор для управления двигателем (сортировочный конвейер вперед)
KM3 – контактор для управления двигателем (нижний конвейер вперед)
KM4 – контактор для управления двигателем (верхний конвейер вперед)
K1 – реле для подачи управляющего сигнала роботу 1 (команда подачи)
K2 – реле для управления сортирующей створкой
K3 – реле для управления толкателем
К4 – реле для подачи управляющего сигнала роботу 2 (команда разгрузить)
Д1…Д3 – датчики определения размера детали
Д4…Д7 – датчики положения
PB1 – кнопка на пульте управления
SW1, SW2 – тумблеры на панели управления
SA1 – кнопка аварийного выключения (кнопка-гриб с фиксацией)

**Контрольные вопросы**

1. Классификация и структура регистров (16-ти и 32-х битных).
2. Перечислите форматы чисел, с которыми могут работать ПЛК.
3. Инструкция передачи данных.
4. Инструкции сравнения числовых данных.
5. Инструкция копирования и инвертирования.
6. Инструкция обмена данными между 2-мя регистрами.
7. Команды сдвига регистра вправо-влево.
8. Синтаксис двоично-десятичного преобразования данных.
9. Синтаксис команды, выполняющей конвертирование «BCD»-данных в двоичный формат.

# ЛАБОРАТОРНАЯ РАБОТА №15

## Разработка программы управления подсветкой рекламного щита

**Цель работы**

1. Получение практических навыков программирования логических контроллеров в среде GX Developer.
2. Разработка программы управления подсветкой рекламного щита в соответствии с заданным алгоритмом.
3. Осуществление диагностики работы программы в *off-line* и *on-line* режимах.

**Краткий обзор пакета GX Developer фирмы Mitsubishi**

Программирование ПЛК может быть выполнено с помощью целого ряда пакетов ПО, соответствующих стандарту МЭК 1131-3 (более подробная информация приведена в главе «Классификация языков по стандарту МЭК 61131-3» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами»). В данной работе для программирования ПЛК используется “родной” пакет ПО, поставляемый фирмой Mitsubishi совместно со своей продукцией – GX Developer.

GX Developer представляет собой стандартное средство программирования контроллеров Mitsubishi семейства FX. Пакет сочетает в себе набор функций разработки, отладки и записи программы в ПЛК с использованием интерфейсных возможностей Windows. Внешний вид среды GX Developer приведен на рисунке 15.1.

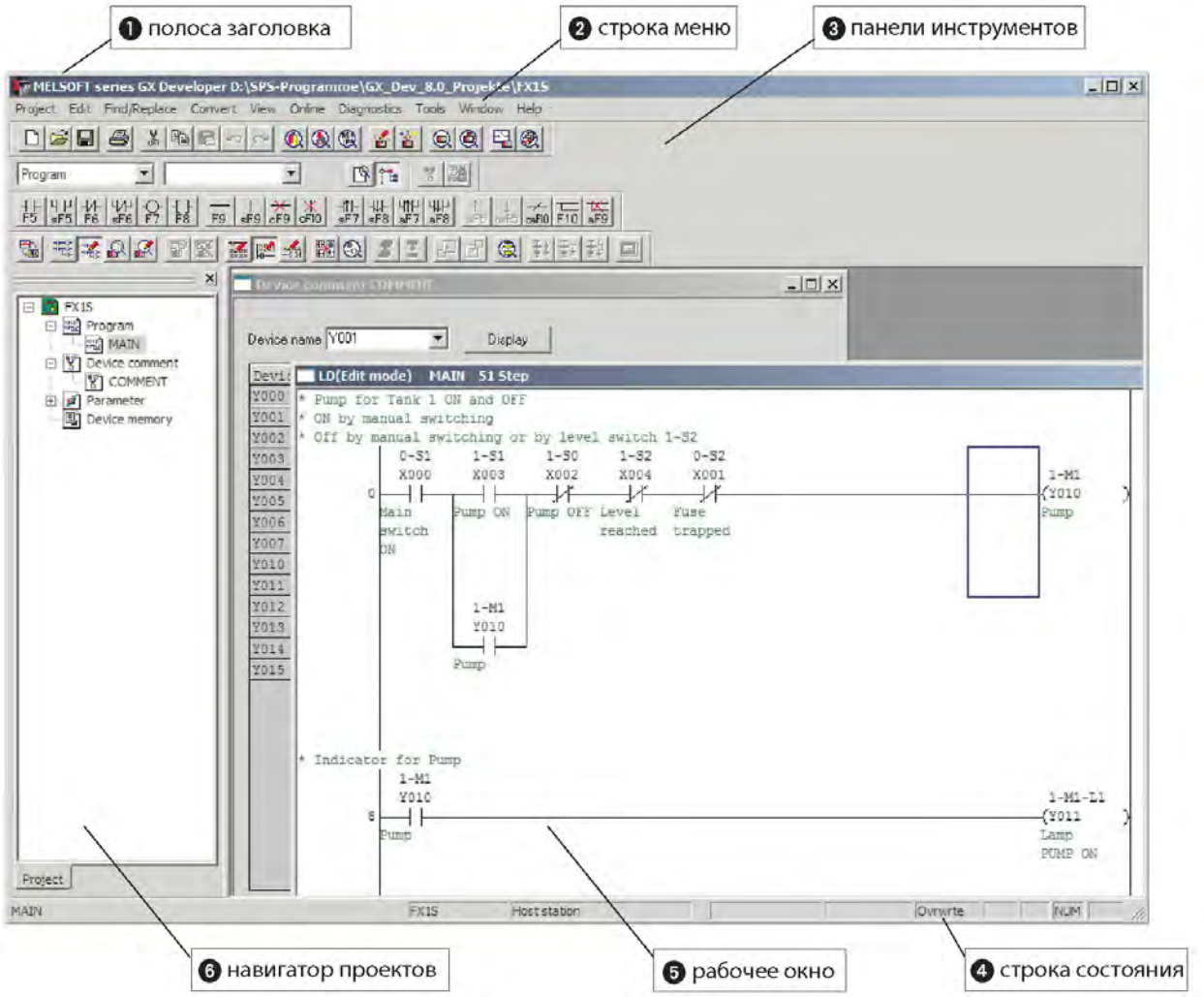
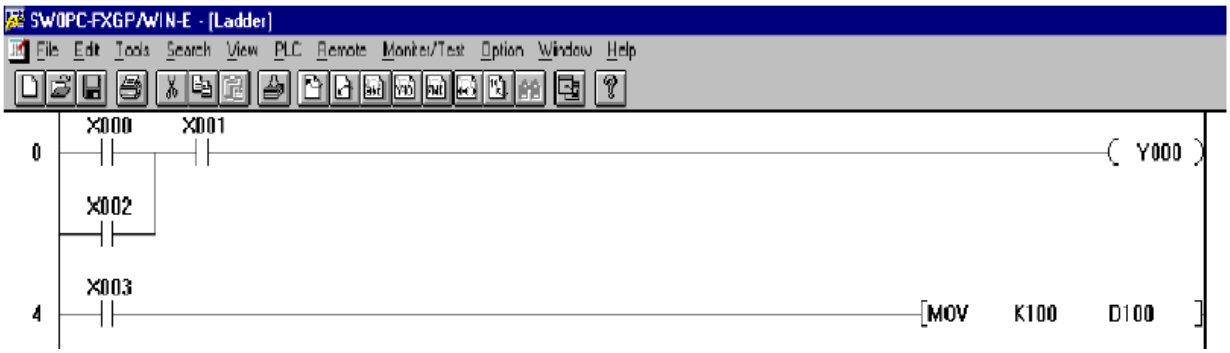



Рис. 15.1. GX Developer. Внешний вид среды разработки программы для ПЛК

Законченные программные модули могут быть скопированы из одного проекта в другой. Также можно копировать строки инструкций в пределах созданной программы. Данный пакет обеспечивает доступ к параметрам контроллеров, конфигурациям сети и таблицам комментариев. Все параметры контроллеров и сетевые установки проверяются на достоверность при входе.

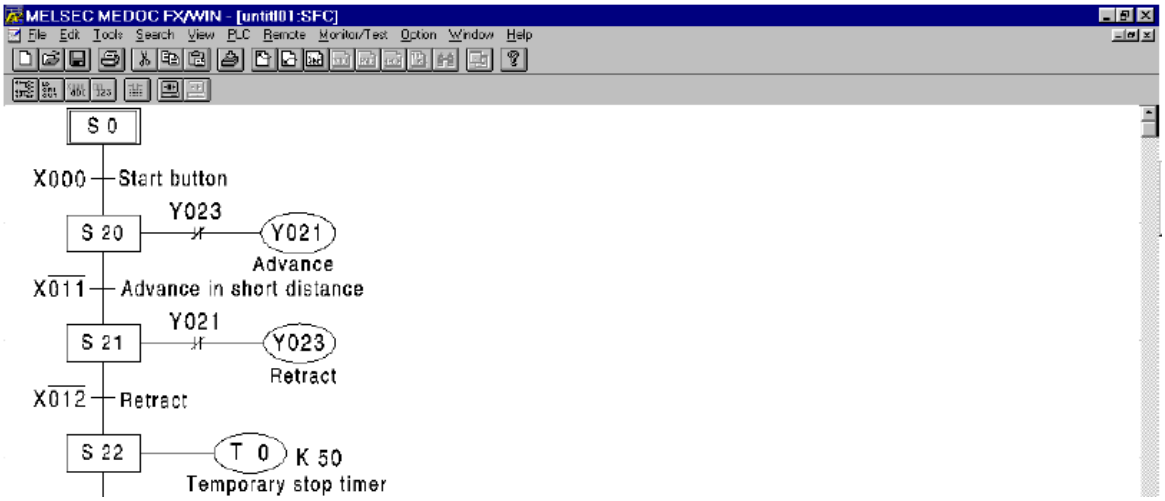
Пакет позволяет использовать при программировании следующие взаимно конвертируемые языки описания приложения:

- ✓ язык релейно-контактных схем MELSEC ladder diagram (LD) - использует в качестве базовых элементов программирования графические элементы "контакты" (contacts) и "катушки" (coils), связанные с входными и выходными каналами соответственно.




- ✓ <u>список инструкций MELSEG instruction list (IL)</u> - унификация интерфейса языка программирования низкого уровня, неориентированного на какую-либо микропроцессорную архитектуру. На его основе можно создавать оптимальные по быстродействию программные единицы.


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




Другой важной особенностью является интеграция программы Ladder Logic Test, которая позволяет программисту производить имитацию работы программы в режиме off-line (см. Рис. 15.2). Интегрированные редакторы *Ladder*

*Logic Test* позволяют манипулировать всеми регистрами данных, вводами-выводами и реле.

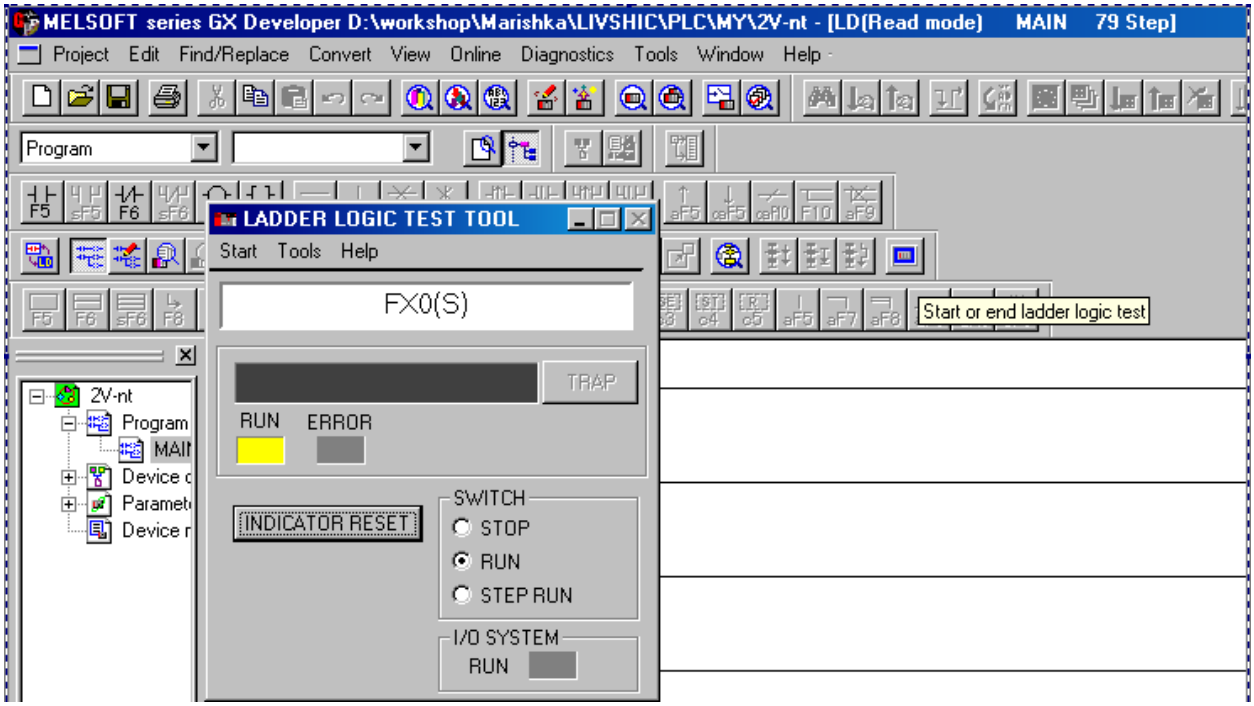


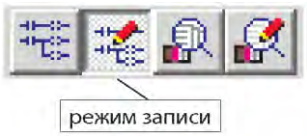

Рис. 15.2. Ladder Logic Test. Внешний вид программы

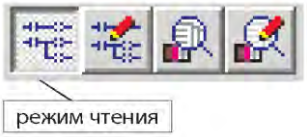
**Краткие сведения по работе с программой**

Пакет GX Developer имеет интуитивно понятный пользовательский интерфейс. Для создания нового проекта следует нажать кнопку *New Project* на панели инструментов и в открывшемся диалоговом окне указать серию и тип программируемого контроллера, а также выбрать язык программирования. Для программирования на языке релейно-контактных схем (LD) в программе предусмотрена отдельная панель инструментов *Ladder Symbols* с подписями «горячих» клавиш.


Переключение между языками описания приложения релейно-контактных схем (LD) и списком инструкций (IL) осуществляется нажатием кнопки *Ladder/Instruction list view switches*.

Для ввода или изменения инструкций должен быть активирован режим записи (*Write mode*). Режим чтения *(Read mode)*, в котором изменение программы невозможно, служит для просмотра программы или поиска операндов.



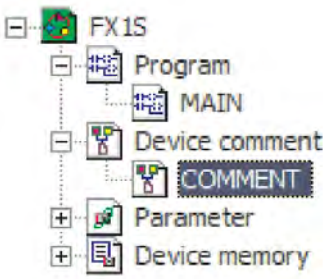

Если в программе указываются только операнды с их адресами, то очень быстро утрачивается обзорность программы. Программы с несколькими сотнями цепей без каких-либо комментариев понимает только разработчик.

Поэтому подробное документирование программы настолько же важно, как и само программирование.

GX Developer предлагает три способа документирования:

- комментарии к операндам
- текстовые вставки *(Statements)*
- надписи

**Комментарий к операнду** - это краткое описание, присвоенное операнду. Комментарии к операндам можно либо обрабатывать в файле COMMENT независимо от программирования (см. Рис. 15.3), либо вводить во время программирования при вводе операнда. В последнем случае файл COMMENT обновляется автоматически. Чтобы комментарии можно было вводить при программировании, необходимо в меню *Tools* открыть окно диалога *Options* и на закладке *Program Common* активировать опцию *Continuous during write*. В этом случае при вводе операнда вы имеете возможность заново ввести комментарий для этого операнда или обработать уже имеющийся комментарий.




Как правило, функция входов и выходов известна уже до программирования, и поэтому комментарии можно ввести заранее, непосредственно в файл комментариев.

Для обработки этого файла щелкните в навигаторе проектов двойным щелчком по COMMENT.

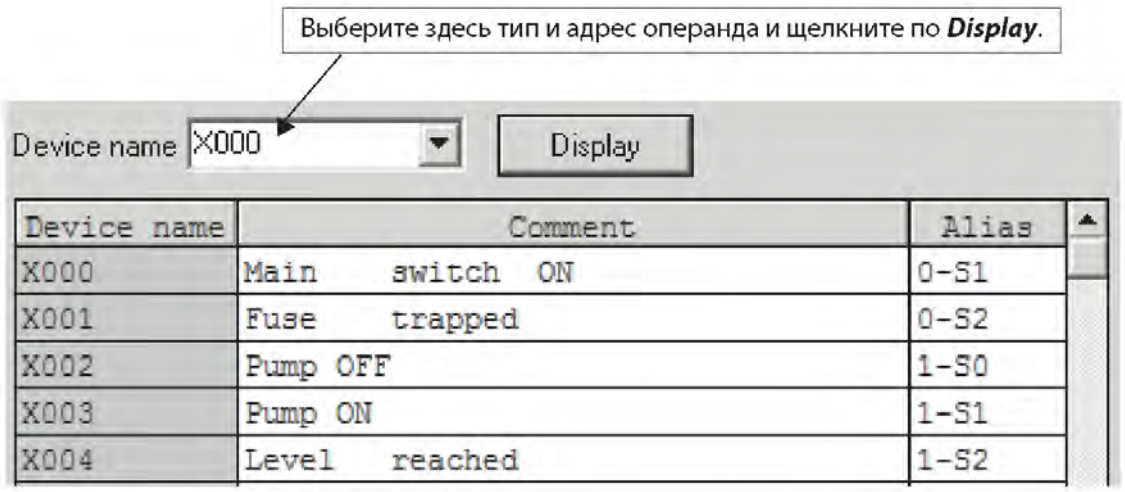



Рис. 15.3. Ввод комментариев к операндам в файл COMMENT

Для каждого операнда можно ввести комментарий *(Comment)* длиной до 32 знаков и альтернативное название *(Alias)* длиной до 8 знаков.

**Альтернативное название** представляет собой краткое обозначение операнда, которое можно показывать на экране вместе с настоящим названием

операнда или вместо него. В качестве альтернативных названий можно использовать, например, идентификационные обозначения конструктивных элементов, по которым можно однозначно идентифицировать входы и выходы установки.

В меню *View* можно выбрать, должны ли в программе показываться комментарии к операндам и (или) альтернативные обозначения.

Кроме того, если комментарии к операндам планируется сохранить в контроллере, то в параметрах контроллера MELSEC семейства FX для этого должна быть зарезервирована память. При этом место, резервируемое для комментариев, отнимается от памяти для программы. Например, в контроллере серии FX1s можно сохранить 2000 шагов программы. Если в нем создается 1 блок для комментариев, как это показано на иллюстрации справа, то в этом контроллере можно сохранить 50 комментариев к операндам. Каждый блок с 50 комментариями уменьшает объем памяти для программы на 500 шагов программы. В контроллере FX1s можно зарезервировать до 3 блоков комментариев с 150 комментариями к операндам. После этого еще остается место для 500 шагов программы. Размер файла с комментариями к операндам можно уменьшить, выполнив в меню *Tools* функцию *Delete unused comments* (стереть неиспользуемые комментарии).

**Текстовые вставки** *Statements* (см. Рис. 15.4) служат для разъяснения и структурирования программы. Они призваны улучшить и ускорить понимание программы. Каждая текстовая вставка изображается в виде одной строки и может содержать до 64 знаков. Для каждой цепи можно ввести 15 строк текстовых вставок.

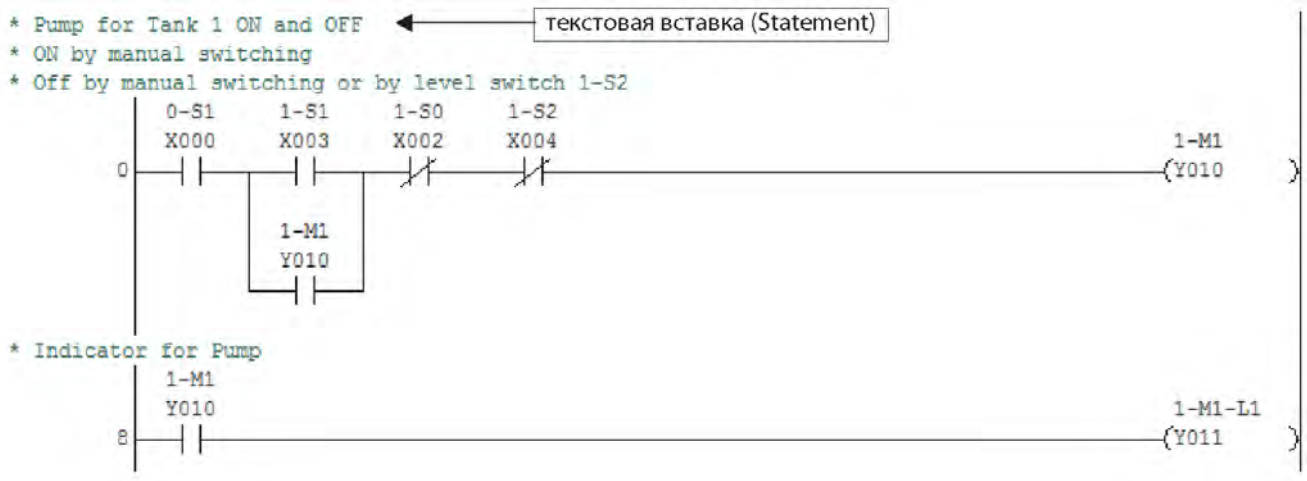



Рис. 15.4. Текстовые вставки в GX Developer

**Надпись** (см. Рис. 15.5) можно сделать для каждой команды вывода или прикладной инструкции в конце цепи тока, в виде одной строки длиной до 32 знаков.

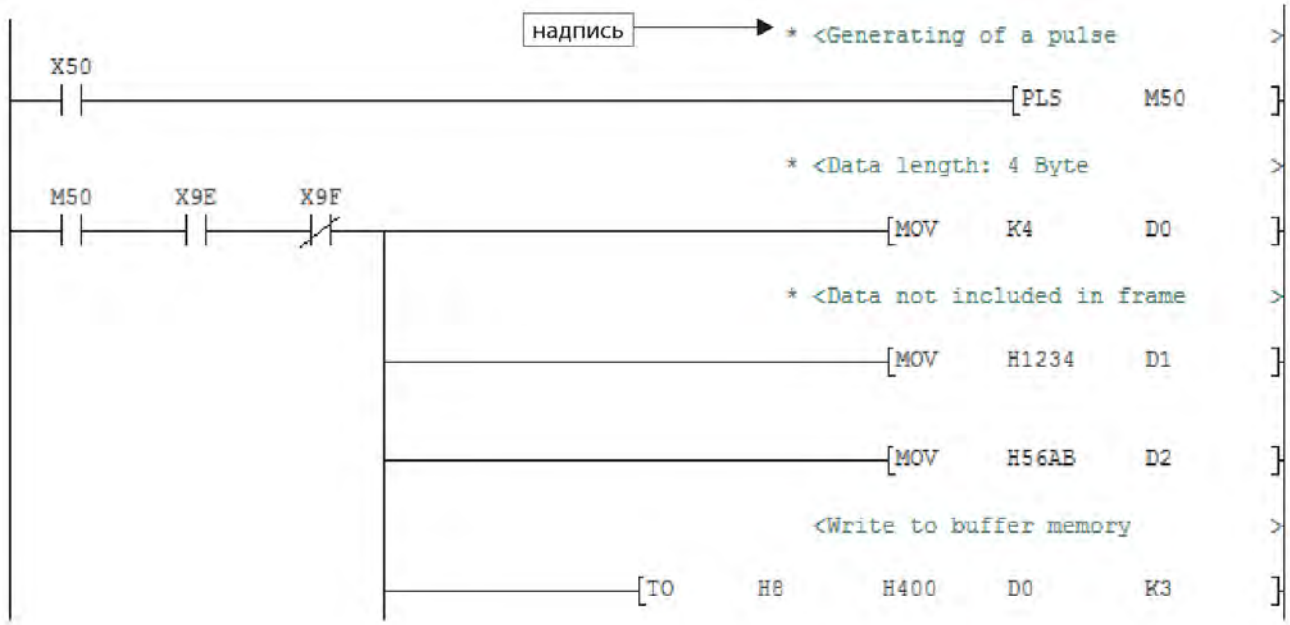



Рис. 15.5. Надписи в GX Developer

Возможность ввода текстовых вставок и надписей можно активировать в меню *Edit –Documentation – Statement* или *Edit – Documentation – Note*. Следует отметить, что запись текстовых вставок и надписей в контроллеры семейства FX невозможна.

Чтобы проанализировать работу программы в режиме off-line, необходимо сначала конвертировать ее в «понятный» контроллеру язык (команда *Convert* в меню *Convert* или клавиша F4). Затем вызывается программа *Ladder Logic Test* нажатием соответствующей кнопки на панели инструментов. Появляется окно запуска программы на выполнение, ее остановки или пошагового выполнения (см. Рис. 15.2) и имитируется запись данных в ПЛК.

Используя монитор переменных *Entry data monitor*, можно наблюдать текущие состояния операндов. Двойным щелчком по пустой ячейке в столбце *Device* (см. Рис. 15.6) вызывается окно регистрации операндов, используемых в программе. Кнопка *Start monitor* в окне *Entry data monitor* запускает программу на выполнение, в ходе которого, следуя заданному алгоритму, выводятся значения зарегистрированных операндов, что дает пользователю максимум прозрачности при анализе работы программы.

При тестировании программы имеется также возможность влиять на состояния или значения операндов непосредственно из программатора. Если, например, для запуска определенного процесса нужен входной сигнал выключателя, то на компьютере этому входу можно присвоить требуемое состояние, а затем наблюдать за дальнейшим ходом программы. Для этого двойным щелчком по операнду, состояние которого нужно задать принудительно, вызывается окно *Device test* (внешний вид его показан на рисунке 15.6), в котором нужно нажать *Force On* для данного элемента.

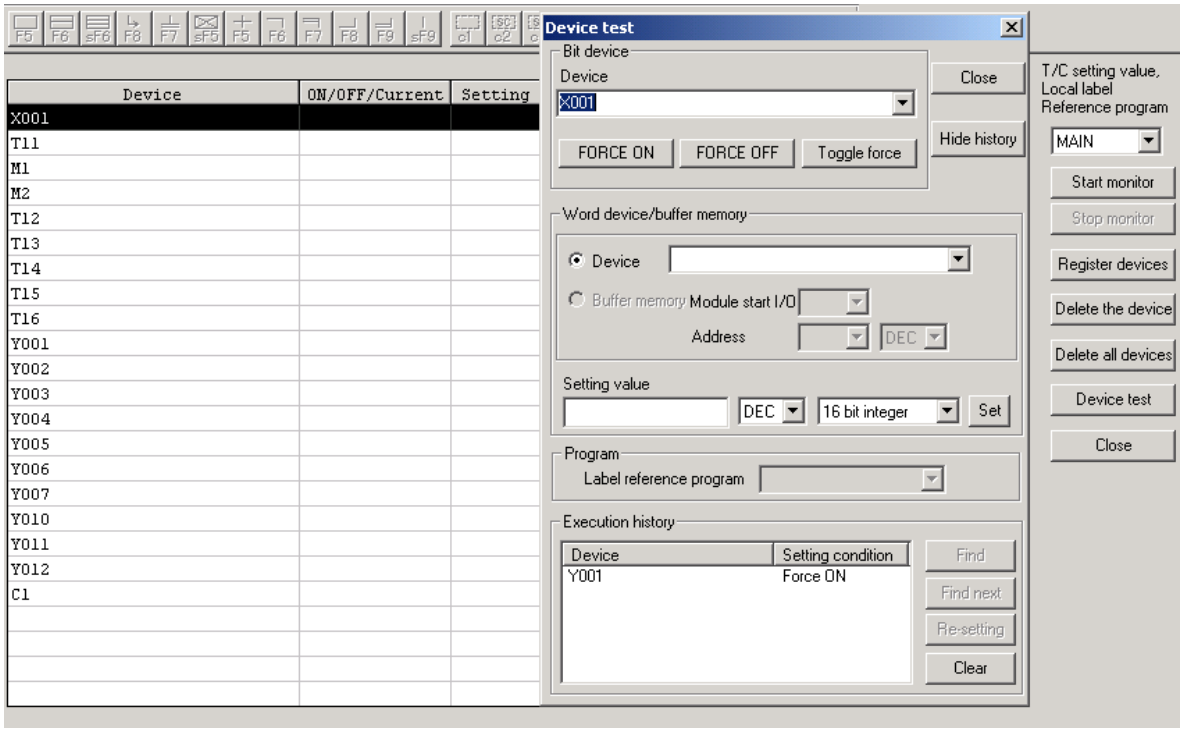
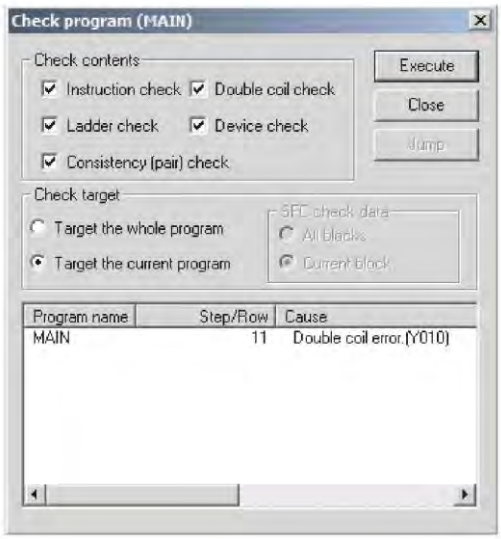



Рис. 15.6. Device test. Внешний вид




Прежде чем записать программу в контроллер, ее необходимо проверить на наличие ошибок. Для этого выберите в меню *Tools* функцию *Check program*. Проверка запускается по нажатию кнопки *Execute*. Результат проверки выводится в нижней части диалогового окна. Переход к месту в программе, где обнаружена ошибка, осуществляется двойным щелчком по соответствующему сообщению об ошибке. В этом примере в качестве операнда команды вывода два раза использован один и тот же выход.

Запись программы в память контроллера осуществляется через последовательные интерфейсы (RS232 – RS422) с помощью специального кабеля подключения через порты Com1 - Com4. Следует отметить, что перед записью программы необходимо перевести переключатель RUN/STOP на контроллере в режим *STOP*, а также стереть из памяти контроллера предыдущую программу нажатием в диалоговом окне *Write to PLC* кнопки *Clear PLC memory*...

Настройка подключения контроллера к программатору вызывается в меню *Online –Transfer setup*.

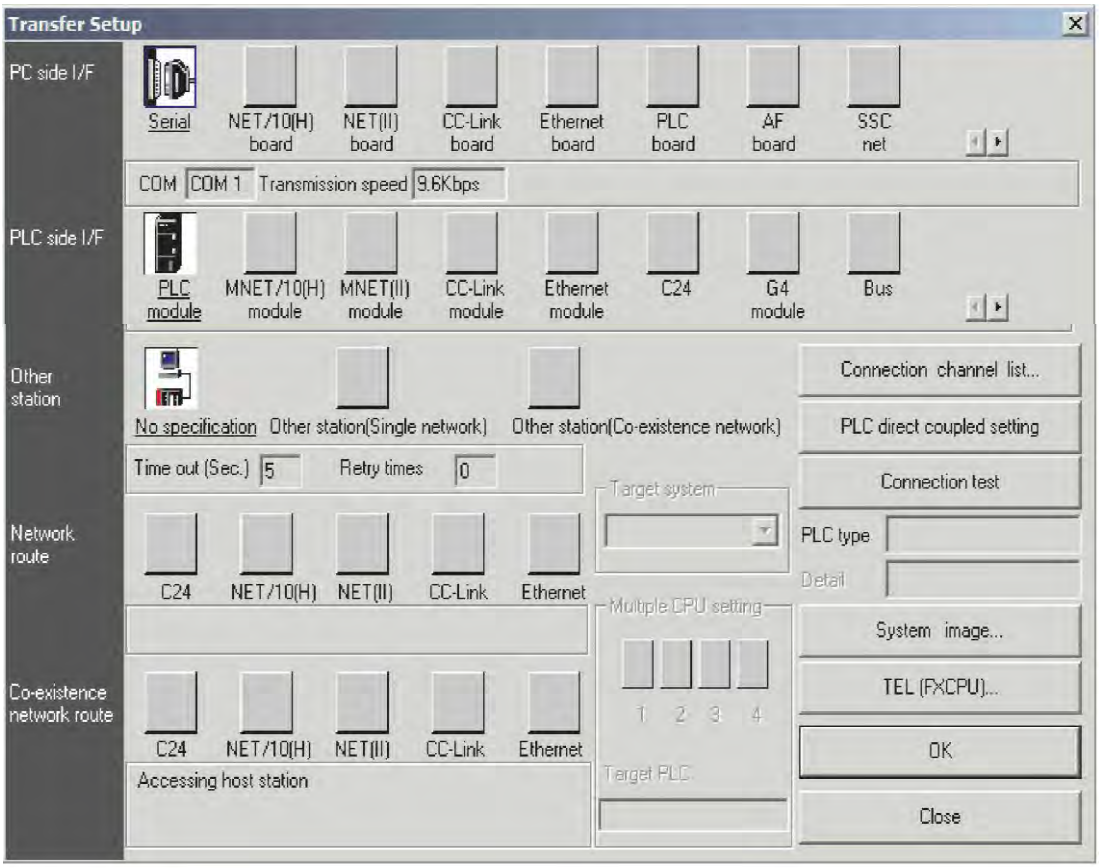
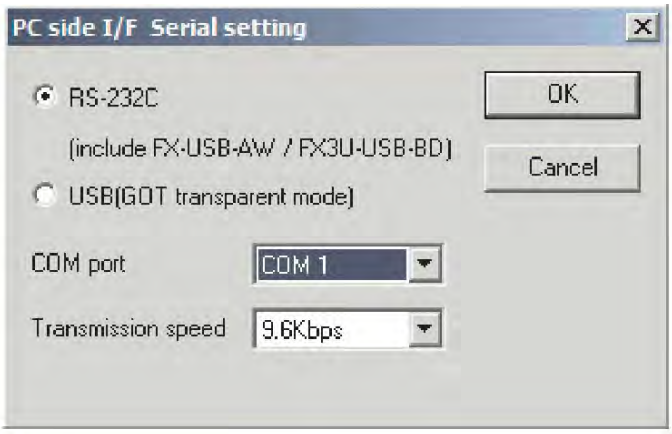



Рис. 15.7. Transfer Setup. Настройка подключения контроллера к программатору




Двойным щелчком по пиктограмме *Serial* вызывается диалоговое окно настройки порта и скорости передачи данных по интерфейсу RS232. Стандартная скорость передачи данных 9,6 кбод. Проверка связи осуществляется по нажатию кнопки *Connection test* в диалоговом окне *Transfer Setup* (см. Рис. 15.7). Если устройства могут обмениваться между собой данными, появляется сообщение: Successfully connected with the FX0sCPU.

Нажатием кнопки *Write to PLC* на стандартной панели инструментов GX Developer вызывается диалоговое окно *Write to PLC*, представленное на рисунке 15.8., в котором необходимо отметить пункт *Program – MAIN.* Нажатие кнопки *Execute* осуществляет запись программы в память контроллера.

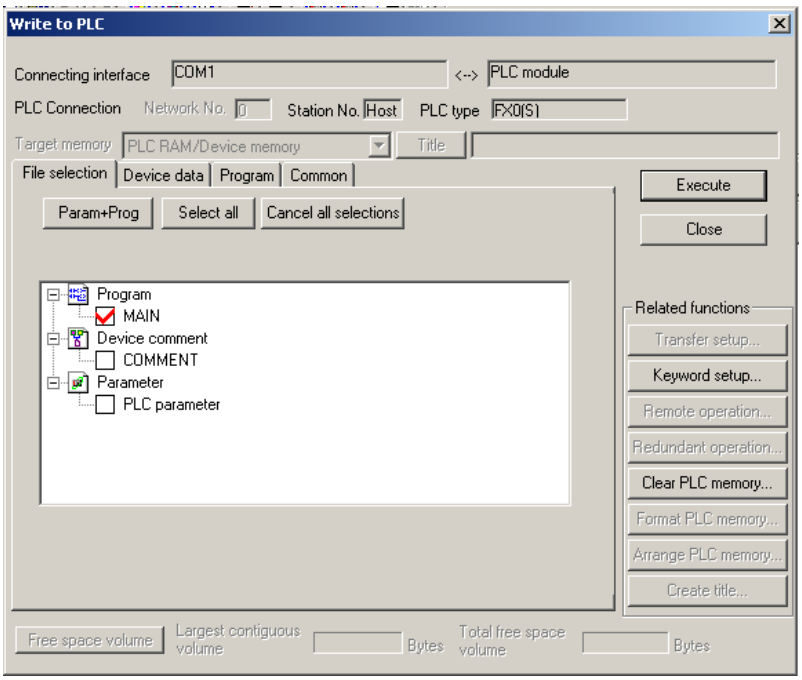



Рис. 15.8. Write to PLC. Запись программы в память контроллера

После выдачи сообщения об успешной записи следует перевести переключатель RUN/STOP на контроллере в режим *RUN*, после чего можно запускать отработку контроллером заданной программы.

**Описание лабораторного стенда**

В лабораторной работе в основе СУ подсветкой рекламного щита заложен программируемый логический контроллер Mitsubishi MELSEC FX0S-30MR-DS. Описание контроллера приведено в таблице 2.1 – Технические характеристики контроллеров главы 2 «Основные характеристики и параметры ПЛК» первой части учебно-методического пособия «Программируемые логические контроллеры для управления технологическими процессами». Внешний вид рекламного щита представлен на рисунке 15.9, схема подключения контроллера – на рисунке 15.10. Каждой букве и логотипу компании Mitsubishi соответствует свой светодиод, подключенный к соответствующим выходам контроллера (Y0…Y12). Справа на щите находится тумблер ON/OFF, подключенный к входу X0 контроллера. В качестве источника питания используется стабилизированный источник питания постоянного тока на 24В.

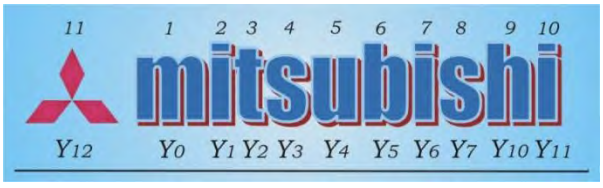



Рис. 15.9. Внешний вид рекламного щита

**Схема подключения контроллера**

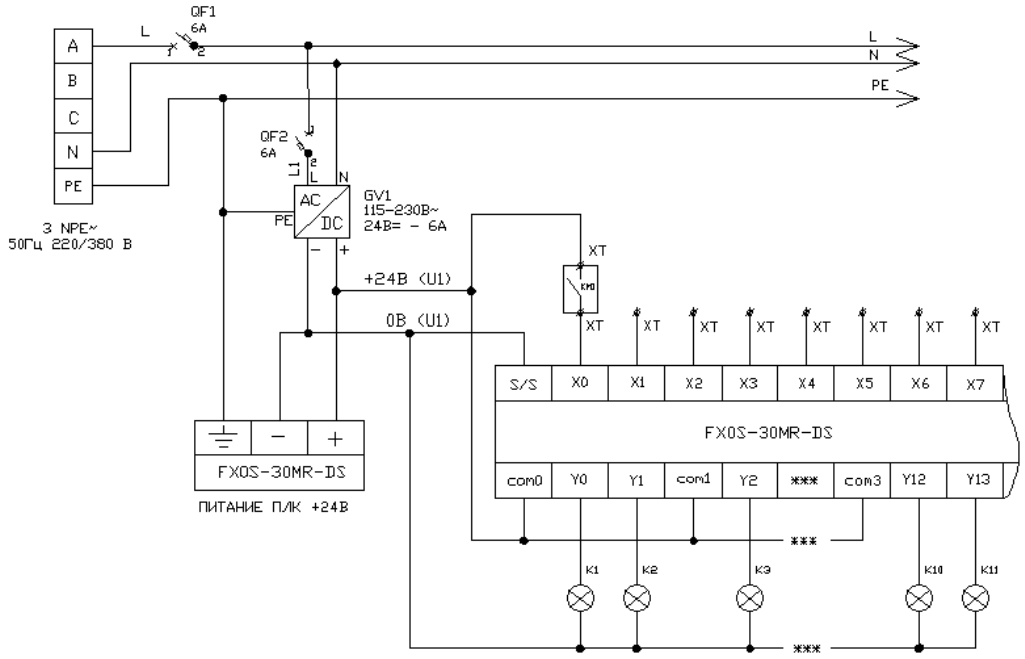



Рис. 15.10. Схема подключения ПЛК

**Задание**

**Вариант 1. Разработка программы управления подсветкой рекламного стенда**

1. Разработать программу, управляющую подсветкой рекламного стенда, по следующему алгоритму:
   1.1 с интервалом 1с после включения тумблера (вход Х0) включаются попарно лампочки 1 и 10 (выходы У0 и У12), 2 и 9 (выходы У1 и У11) и т.д.;
   1.2 после включения последней пары лампочек 5 и 6 (выходы У4 и У5) и выдерживается пауза 1с;
   1.3 после паузы включается лампочка 11 под логотипом (выход У12);
   1.4 через 1с все лампочки одновременно отключаются на 0,5с;
   1.5 через 0,5с все лампочки одновременно включаются и выдерживается пауза 0,5с;
   1.6 повторить пункты 1.4 и 1.5 еще 2 раза;
   1.7 погасить одновременно все лампочки на 1с;
   1.8 продолжить отработку программой заданного алгоритма, начиная с п.1.1.
2. Произвести имитацию работы программы в режиме off-line посредством программы Ladder Logic Test.
3. Убедившись в выполнении программой заданного алгоритма, записать ее в память контроллера и продемонстрировать работу на стенде.

Более наглядно последовательность включения-выключения лампочек представлена в 15.1

Таблица 15.1

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | Пауза |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Тумблер X0 | | | | | | | | | | | 1с |
| 1.1 | 1 | - | - | - | - | - | - | - | - | 10 | - | 1с |
| | 1 | 2 | - | - | - | - | - | - | 9 | 10 | - | 1с |
| | 1 | 2 | 3 | - | - | - | - | 8 | 9 | 10 | - | 1с |
| | 1 | 2 | 3 | 4 | - | - | 7 | 8 | 9 | 10 | - | 1с |
| 1.2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| 1.3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1с |
| 1.4 | - | - | - | - | - | - | - | - | - | - | - | 0,5с |
| 1.5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 0,5с |
| 1.6 | Возврат к п.1.4 (2 раза) | | | | | | | | | | | |
| 1.7 | - | - | - | - | - | - | - | - | - | - | - | 1с |
| 1.8 | Возврат к п.1.1 | | | | | | | | | | | |

**Вариант 2. Разработка программы управления подсветкой рекламного стенда**

1. Разработать программу, управляющую подсветкой рекламного стенда, по следующему алгоритму:
   1.1 с интервалом 1с после включения тумблера (вход X0) включается лампочка 11 под логотипом (выход У12);
   1.2 через 1 с включаются поочередно лампочки 1, 2 и 3 (выходы У0, У1 и У2);
   1.3 через 1с после включения лампочки 3 одновременно включается лампочка 4 (выход У3) и отключается лампочка 1 (выход У0);
   через 1с после включения лампочки 4 одновременно включается лампочка 5 (выход У4) и отключается лампочка 2 и т.д.;
   1.4 с интервалом 1с после одновременного включения последней лампочки 10 (выход У11) и отключения лампочки 7 (выход У6) отключаются лампочки 8, 9, 10 (выходы У7, У10 и У11) и лампочка 11 под логотипом (выход У12);
   1.5 выдерживается пауза 1с и одновременно на 1с включаются все лампочки;
   1.6 отключается лампочка 11 под логотипом (У12). Выдерживается пауза 1с;
   1.7 через 1с одновременно отключаются лампочки 1, 2 и 9, 10. Выдерживается пауза 1с;
   1.8 одновременно отключаются лампочки 3, 4 и 7, 8. Выдерживается пауза 1с;
   1.9 одновременно отключаются лампочки 5 и 6;
   1.10 повторить пункты 1.5 – 1.9 еще 2 раза;
   1.11 продолжить отработку программой заданного алгоритма, начиная с п.1.1.

2. Произвести имитацию работы программы в режиме off-line посредством программы Ladder Logic Test.
3. Убедившись в выполнении программой заданного алгоритма, записать ее в память контроллера и продемонстрировать работу на стенде.

Более наглядно последовательность включения-выключения лампочек представлена в таблице 15.2

Таблица 15.2

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | Пауза |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Тумблер X0 | | | | | | | | | | | 1с |
| 1.1 | - | - | - | - | - | - | - | - | - | - | 11 | |
| 1.2 | 1 | - | - | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | - | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | - | - | - | - | - | - | - | 11 | 1с |
| 1.3 | - | 2 | 3 | 4 | - | - | - | - | - | - | 11 | 1с |
| | - | - | 3 | 4 | 5 | - | - | - | - | - | 11 | 1с |
| | - | - | - | 4 | 5 | 6 | - | - | - | - | 11 | 1с |
| | - | - | - | - | 5 | 6 | 7 | - | - | - | 11 | 1с |
| | - | - | - | - | - | 6 | 7 | 8 | - | - | 11 | 1с |
| | - | - | - | - | - | - | 7 | 8 | 9 | - | 11 | 1с |
| | - | - | - | - | - | - | - | 8 | 9 | 10 | 11 | 1с |
| 1.4 | - | - | - | - | - | - | - | - | 9 | 10 | 11 | 1с |
| | - | - | - | - | - | - | - | - | - | 10 | 11 | 1с |
| | - | - | - | - | - | - | - | - | - | - | 11 | 1с |
| | - | - | - | - | - | - | - | - | - | - | - | 1с |
| 1.5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1с |
| 1.6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| 1.7 | - | - | 3 | 4 | 5 | 6 | 7 | 8 | - | - | - | 1с |
| 1.8 | - | - | - | - | 5 | 6 | - | - | - | - | - | 1с |
| 1.9 | - | - | - | - | - | - | - | - | - | - | - | 1с |
| 1.10 | Возврат к п.1.5 (2 раза) | | | | | | | | | | | |
| 1.11 | Возврат к п.1.1 | | | | | | | | | | | |

**Вариант 3. Разработка программы управления подсветкой рекламного стенда**

1. Разработать программу, управляющую подсветкой рекламного стенда, по следующему алгоритму:
   1.1 с интервалом 1с после включения тумблера (вход X0) включаются поочередно лампочки 11, 1, 2, …10 (выходы У12, У0, У1, … У11);
   1.2 через 1с после включения последней лампочки 10 поочередно отключаются лампочки 11, 1, 2, …10 с интервалом 1с;
   1.3 через 1с после отключения лампочки 10 попарно включаются лампочки 1 и 10, 2 и 9, 3 и 8, 4 и 7, 5 и 6 с интервалом 1с;
   1.4 выдерживается пауза 1с и все лампочки одновременно отключаются;

1.5 повторить пункты 1.3 и 1.4 еще 2 раза;

1.6 продолжить отработку программой заданного алгоритма, начиная с п.1.1.

2. Произвести имитацию работы программы в режиме off-line посредством программы Ladder Logic Test.

3. Убедившись в выполнении программой заданного алгоритма, записать ее в память контроллера и продемонстрировать работу на стенде.

Более наглядно последовательность включения-выключения лампочек представлена в таблице 15.3

Таблица 15.3

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | Пауза |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Тумблер X0 | | | | | | | | | | | 1с |
| 1.1 | - | - | - | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | - | - | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | - | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | - | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | - | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | - | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | - | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | - | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | - | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | - | 11 | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1с |
| 1.2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | - | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | - | - | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | - | - | - | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | - | - | - | - | 6 | 7 | 8 | 9 | 10 | - | 1с |
| | - | - | - | - | - | - | - | 8 | 9 | 10 | - | 1с |
| | - | - | - | - | - | - | - | - | - | 10 | - | 1с |
| | - | - | - | - | - | - | - | - | - | - | - | 1с |
| 1.3 | 1 | - | - | - | - | - | - | - | - | 10 | - | 1с |
| | 1 | 2 | - | - | - | - | - | - | 9 | 10 | - | 1с |
| | 1 | 2 | 3 | - | - | - | - | 8 | 9 | 10 | - | 1с |
| | 1 | 2 | 3 | 4 | - | - | 7 | 8 | 9 | 10 | - | 1с |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | - | 1с |
| 1.4 | - | - | - | - | - | - | - | - | - | - | - | 1с |
| 1.5 | Возврат к п.1.3 (2 раза) | | | | | | | | | | | |
| 1.6 | Возврат к п.1.1 | | | | | | | | | | | |

**Контрольные вопросы**

1. Какой интерфейс Вы использовали в лабораторной работе для программирования контроллера Melsec FX0s?
2. Какие взаимно конвертируемые языки программирования контроллеров поддерживает пакет GX Developer?
3. Для чего предназначена интегрированная в GX Developer программа Ladder Logic Test?
4. В чем заключаются основные достоинства и недостатки визуальных языков программирования ПЛК (FBD, SFC, LD) по сравнению с текстовыми (ST, IL)?
5. В каком состоянии центрального процессора контроллера осуществляется передача данных в ПЛК?

# ЛАБОРАТОРНАЯ РАБОТА №16

## Разработка программного обеспечения для управления шаговым двигателем

**Цель работы**

1. Управление шаговым двигателем от электронной схемы и от контроллера.
2. Разработка программы управления шаговым двигателем от контроллера в соответствии с заданными условиями в пакете GX Developer.

**Краткие теоретические сведения**

Шаговый двигатель – это электромеханическое устройство, которое преобразует электрические импульсы в дискретные механические перемещения. Шаговые двигатели (ШД) являются одним из перспективных типов исполнительных двигателей для систем автоматизации. К достоинствам ШД можно отнести возможность осуществлять точное позиционирование и регулировку скорости без датчика обратной связи, возможность непосредственного цифрового управления от ПК или специализированного контроллера, высокое быстродействие и большой вращающий момент.

**Основные особенности ШД:**

– угол поворота ротора определяется числом импульсов, которые поданы на обмотки управления двигателя;

– скорость вращения вала пропорциональна частоте управляющих импульсов;

– однозначная зависимость положения от входных импульсов обеспечивает позиционирование без обратной связи;

– двигатель обеспечивает полный момент в состоянии останова, если обмотки запитаны;

– прецизионное позиционирование; хорошие ШД имеют точность 3-5% от величины шага. Эта ошибка не накапливается от шага к шагу;

– возможность быстрого старта, остановки, реверсирования;

– высокая надежность, связанная с отсутствием щеток, срок службы шагового двигателя фактически определяется сроком службы подшипников.

Существуют три основных типа ШД:

– двигатели с переменным магнитным сопротивлением;

– двигатели с постоянными магнитами;

– гибридные двигатели.

Большинство современных ШД являются гибридными. Гибридный двигатель является двигателем с постоянными магнитами, но с большим числом полюсов. По способу управления все двигатели одинаковы. Чаще всего на практике двигатели имеют 100 или 200 шагов на оборот, соответственно шаг равен 3,6º или 1,8º.

В зависимости от конфигурации обмоток ШД делятся на биполярные и униполярные. Биполярный двигатель имеет одну обмотку в каждой фазе, которая для изменения направления магнитного поля должна переполюсовываться управляющим контроллером. Всего биполярный двигатель имеет две обмотки и, соответственно, четыре вывода (см. Рис. 16.1а). Униполярный двигатель также имеет одну обмотку в каждой фазе, но от середины обмотки сделан отвод. Это позволяет изменять направление магнитного поля, создаваемого обмоткой, простым переключением половин обмотки (см. Рис. 16.1б). При этом существенно упрощается схема системы управления. Иногда униполярные двигатели имеют раздельные 4 обмотки. Каждая обмотка имеет отдельные выводы, поэтому всего выводов 8 (см. Рис. 16.1в). При соответствующем соединении обмоток такой двигатель можно использовать как униполярный или как биполярный.

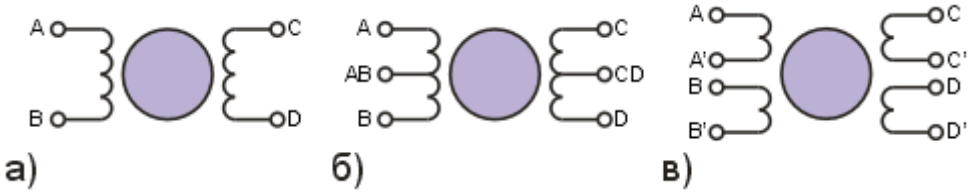



Рис. 16.1 Биполярный ШД (а), униполярный (б) и четырехобмоточный (в)

Существует несколько способов управления фазами шагового двигателя. Первый способ обеспечивается попеременной коммутацией фаз, при этом они не перекрываются, в один момент времени включена только одна фаза (см. Рис. 16.2а). Этот способ называют "one phase on" full step. Точки равновесия ротора для каждого шага совпадают с "естественными" точками равновесия ротора у незапитанного двигателя. Недостатком этого способа управления является то, что для биполярного двигателя в один и тот же момент времени используется 50% обмоток, а для униполярного – только 25%. Это означает, что в таком режиме не может быть получен полный момент.

Второй способ - управление фазами с перекрытием: две фазы включены в одно и то же время. Его называют "two-phase-on" full step. При этом способе управления ротор фиксируется в промежуточных позициях между полюсами статора (см. Рис. 16.2б) и обеспечивается примерно на 40% больший момент, чем в случае одной включенной фазы. Этот способ управления обеспечивает такой же угол шага, как и первый способ, но положение точек равновесия ротора смещено на половину шага.

Третий способ является комбинацией первых двух и называется полушаговым режимом, "one and two-phase-on" half step, когда двигатель делает шаг в половину основного. Каждый второй шаг запитана лишь одна фаза, а в остальных случаях запитаны две (см. Рис. 16.2в). В результате угловое перемещение ротора составляет половину угла шага для первых двух способов управления.

Еще один способ управления называется микрошаговым режимом или micro stepping mode. При этом способе управления ток в фазах нужно менять небольшими шагами, обеспечивая таким образом дробление половинного шага на еще меньшие микрошаги. Для реализации микрошагового режима требуются значительно более сложный управляющий контроллер, позволяющий задавать ток в обмотках с необходимой дискретностью. Полушаговый режим является частным случаем микрошагового режима, но он не требует формирования ступенчатого тока питания катушек, поэтому часто реализуется.

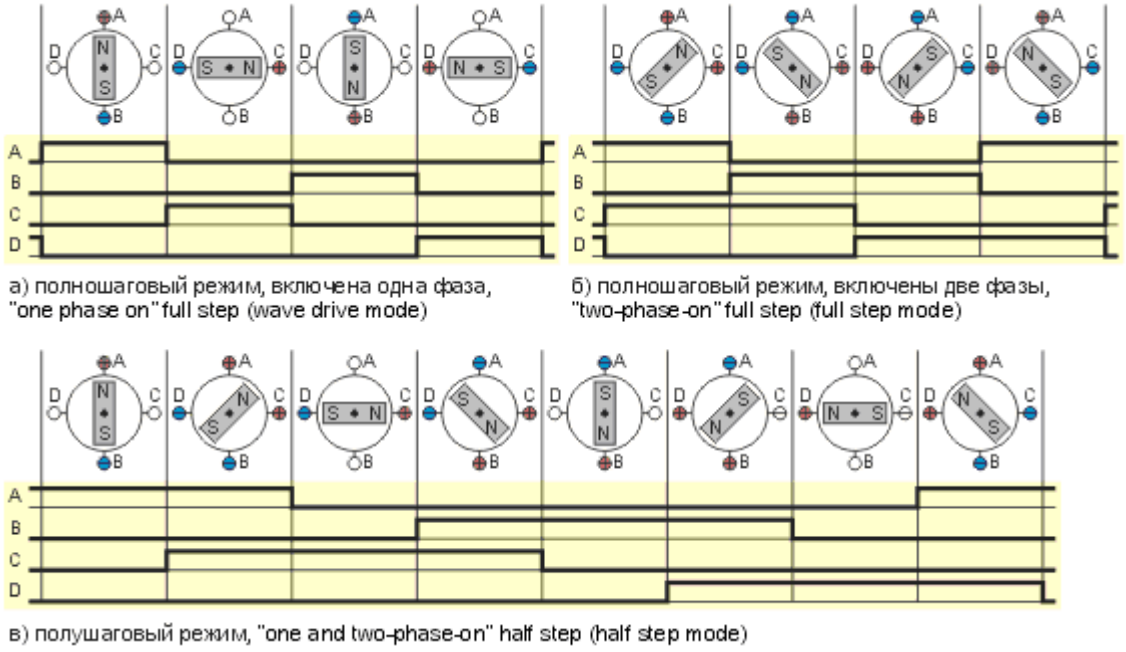



Рис. 16.2 – Различные способы управления фазами шагового двигателя

Лабораторный стенд реализует полношаговый режим управления фазами ШД с одновременно включенными двумя фазами «two-phase-on" full step mode. Переключение управления ШД от электронной схемы к контроллеру осуществляется ключом.

Электронная схема системы управления шаговым двигателем решает две основные задачи: это формирование необходимых временных последовательностей сигналов и обеспечение необходимого тока в обмотках. На практике можно обойтись и без электронных схем. Например, функцию формирования необходимых временных последовательностей сигналов можно реализовать программно, а для обеспечения необходимого тока в обмотках применить набор дискретных транзисторов. Однако при этом схема управления может получиться слишком громоздкой. Несмотря на это, в некоторых случаях такое решение будет экономически выгодным.

На рисунке 16.3 приведена функциональная схема управления шаговым двигателем.

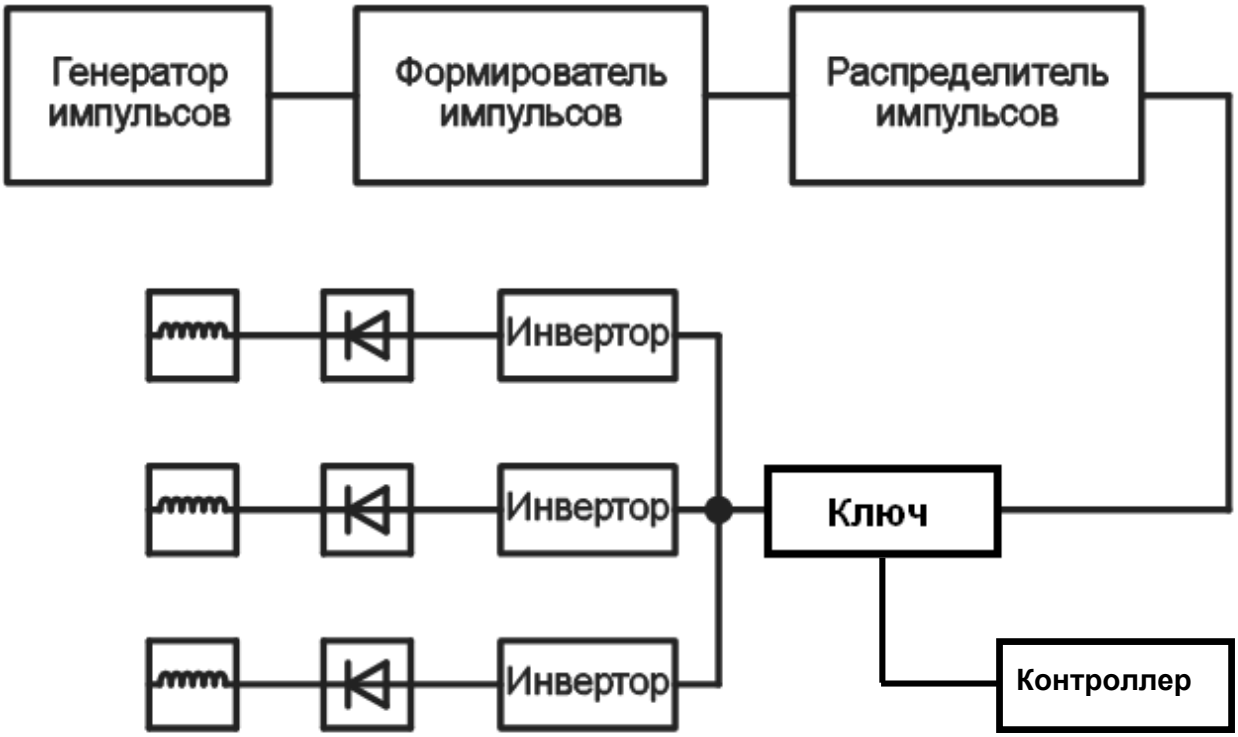



Рис. 16.3 – Функциональная схема управления шаговым двигателем

Функциональная схема привода шагового двигателя состоит из генератора импульсов (ГИ), который может быть представлен готовым устройством (микросхемой) либо выходом с какой-либо схемы управления работой двигателя. В любом случае этот блок отвечает за подачу на двигатель импульсов регулируемой частоты напряжением 0,5В.

Импульсы с выхода ГИ поступают на формирователь импульсов (ФИ), поступающих на синхронизирующий вход распределителя импульсов.

Распределитель импульсов (РИ) непосредственно отвечает за управление обмотками шагового двигателя (ШД). РИ позволяет получить на выходе код, соответствующий подаче напряжения на обмотки ШД.

Ключ используется для переключения управления ШД от микросхемы к контроллеру. Инвертор предназначен для управления формированием вектора магнитного потока одновременно на двух из трех статорных обмотках ШД. Диоды необходимы для защиты схемы от обратных токов, возникающих под действием ЭДС самоиндукции в момент коммутации обмоток ШД.

Электрическая схема электронного блока системы управления ШД представлена на рисунке 16.5.

Импульсы с выхода ГИ поступают на формирователь импульсов DD1, представляющий собой мультивибратор на микросхеме К155 ТЛ1. С выхода мультивибратора импульсы поступают на логическую схему распределения импульсов DD2, состоящую из двух D-триггеров с динамическим управлением и 4-х элементов 2-И-НЕ.

Напомним, что запись информации в D-триггер с динамическим управлением происходит только в момент перехода тактового сигнала на

синхронизирующем входе С из 0 в 1. При постоянном значении С=0, С=1 или отрицательном перепаде триггер хранит предыдущую информацию. На рисунке 16.4 показана временная диаграмма работы D-триггера с динамическим управлением в общем случае.

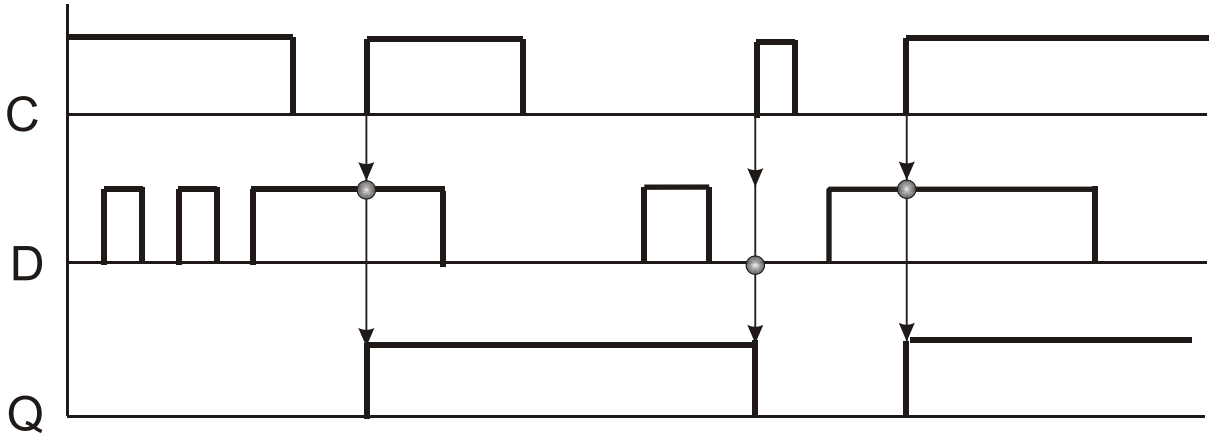



Рис. 16.4 – Временная диаграмма работы D-триггера

Рассмотрим работу схемы распределения импульсов DD2. С инверсных выходов DD2.1 и 2.2 (6 и 9 ножки) поступает сигнал высокого уровня на 3 элемента 2-И-НЕ (DD3). В итоге получаем сигнал низкого уровня на ножках 3, 6 и 8 микросхемы DD3 и сигнал высокого уровня на ножке 11 микросхемы DD3. Это значит, что на базы транзисторов VT1–VT3 поступают соответственно «0», «1», «0». Сигнал высокого уровня с ножки 11 микросхемы DD3 поступает на информационный вход D триггера DD2.1, а сигнал низкого уровня с прямого выхода DD2.1 (ножка 5) поступает на информационный вход D триггера DD2.2.

При поступлении первого синхроимпульса с выхода мультивибратора на входы синхронизации D-триггеров происходит запоминание сигналов с информационных входов D – на прямом выходе триггера DD2.2 – «0», на инверсном – «1», на прямом выходе триггера DD2.1 – «1», на инверсном – «0». Соответственно изменят свои выходные сигналы элементы 2-И-НЕ. На базы транзисторов VT1–VT3 поступают соответственно «1», «0», «0». Сигнал низкого уровня с ножки 11 микросхемы DD3 поступает на информационный вход D триггера DD2.1, а сигнал высокого уровня с прямого выхода DD2.1 (ножка 5) поступает на информационный вход D триггера DD2.2.

При поступлении второго синхроимпульса с мультивибратора происходит запоминание сигналов с информационных входов – на прямом выходе триггера DD2.2 – «1», на инверсном – «0», на прямом выходе триггера DD2.1 – «0», на инверсном – «1». Соответственно изменят свои выходные сигналы элементы 2-И-НЕ. На базы транзисторов VT1–VT3 поступают соответственно «0», «0», «1».

С приходом третьего синхроимпульса происходит запоминание сигналов с информационных входов – на прямом выходе триггера DD2.2 – «0», на инверсном – «1», на прямом выходе триггера DD2.1 – «0», на инверсном – «1».

Соответственно изменят свои выходные сигналы элементы 2-И-НЕ. На базы транзисторов VT1–VT3 поступают соответственно «0», «1», «0».

Транзисторы VT1 – VT3 выполняют функцию ключей. При поступлении высокого уровня напряжения на базу транзистора происходит его открытие и на соответствующей линии, а следовательно и на обмотке двигателя, напряжение будет равно 0. При низком уровне напряжения на базе транзистора последний будет закрыт, и все напряжение будет подаваться на соответствующую обмотку двигателя.

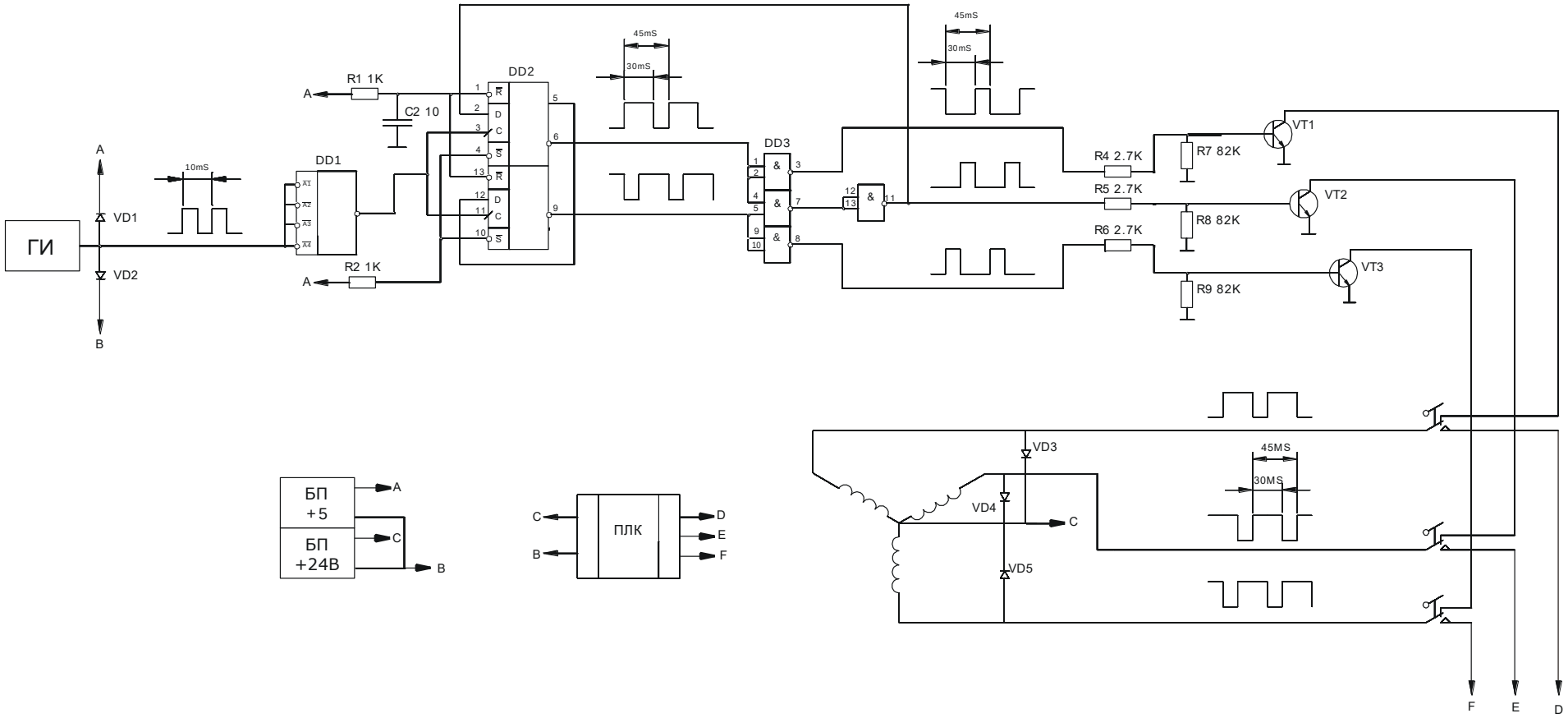



Рис. 16.5 – Схема управления шаговым двигателем

Как видим, в любой момент времени напряжение будет подано на 2 из 3 обмоток двигателя, причем обмотки, переключаясь, заставят двигатель вращаться с частотой в 3 раза меньшей, чем частота входных импульсов.

Временные диаграммы работы схемы при частоте 200 Гц показаны на рисунке 16.6.

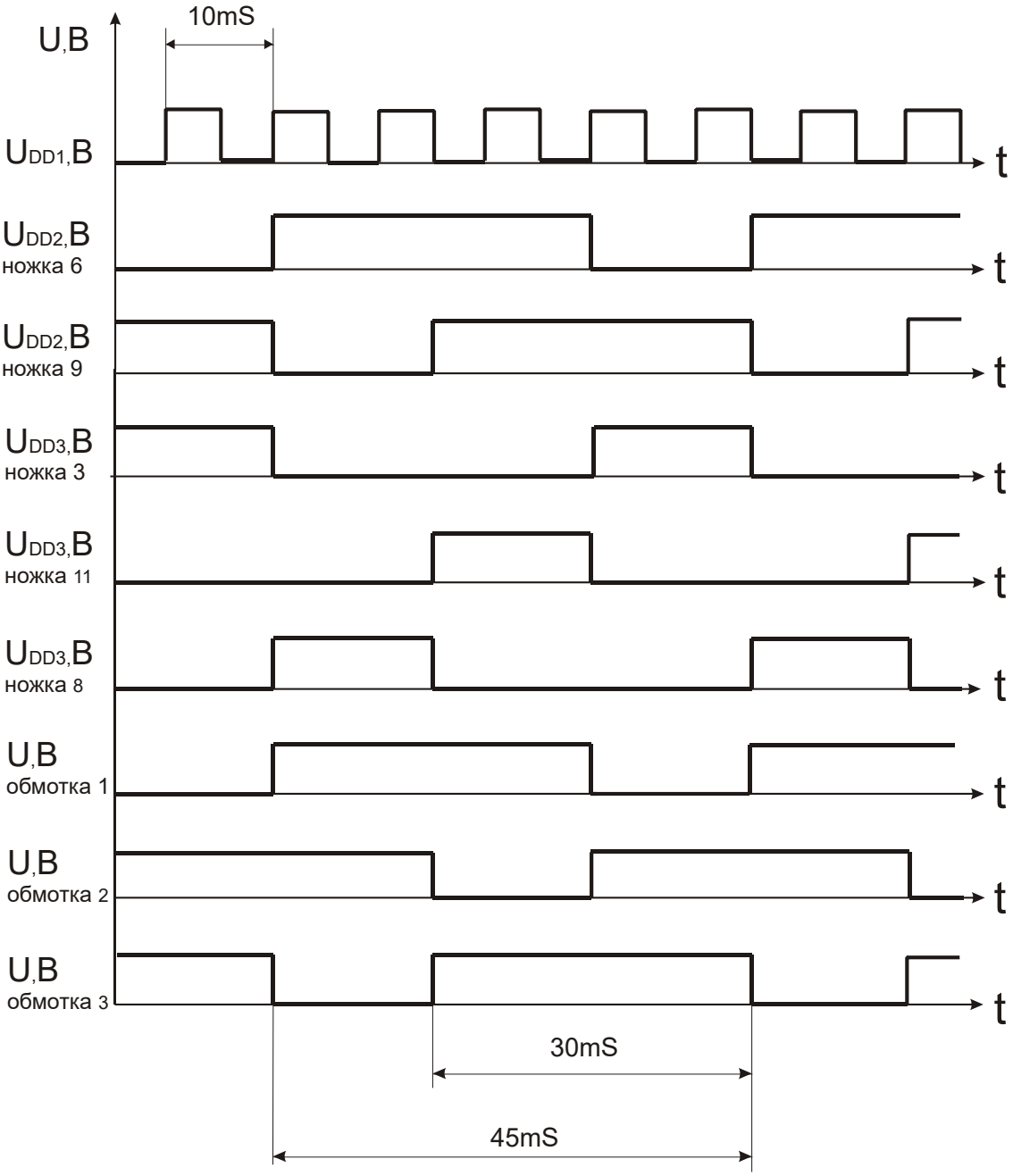



Рис. 16.6 – Временные диаграммы работы схемы при частоте 200 Гц

Первый график отображает вид импульсов на выходе распределителя импульсов DD1. Второй и третий графики отображают вид импульсов на ножках 6 и 9 микросхемы DD2. Следующие три графика отображают импульсы на выходе микросхемы DD3 на ножках 3, 11 и 8 соответственно. Последние 3 графика отображают импульсы на выходе транзисторов VT1 – VT3.

Предложенная в лабораторной работе схема управления шаговым двигателем питается напряжением 5 В от универсального источника питания ИПГ8В, двигатель и контролер питаются напряжением 24 В от источника питания постоянного тока Б5-49.

Частота управления ШД лежит в диапазоне 170-230 Гц.

**Задание**

1. Изучить систему управления ШД на базе электронного блока и получить навыки работы с шаговым приводом, изменяя частоту управляющих импульсов от генератора.
2. Используя временные диаграммы работы схемы управления ШД (см. Рис. 16.6), разработать управляющую программу ШД с учетом следующих условий:
   – режим управления фазами ШД – полношаговый с одновременно включенными двумя фазами «two-phase-on" full step mode;
   – длительность импульсов, подаваемых на обмотку ШД 30мс;
   – длительность паузы 15 мс.

   Для задания требуемой длительности импульсов и пауз необходимо использовать специальные таймеры T32–T55 с таймерным временем дискретизации 10ms, тогда как у таймеров T0–T31, которые использовались в предыдущих лабораторных работах, время дискретизации 100ms. Активация таймеров T32-T55 осуществляется при включенном специальном меркере M8028. Теоретические сведения о специальных меркерах приведены в разделе 11 «Язык релейно-контактных схем (LD)» главе 2 «Программирование внутреннего реле», а также в разделе 12 «Инструкции процесса отработки программы» главе 4 «Ввод прерывания программы (IRET, EI, DI)» первой части учебно-методического пособия [23].
3. Произвести тестирование работы управляющей программы в режиме off-line посредством программы Ladder Logic Test.
4. Убедившись в выполнении программой заданных условий, записать ее в память контроллера и продемонстрировать работу ШД на стенде.
5. Реализовать программно реверс направления вращения двигателя.

**Контрольные вопросы**

1. Опишите известные Вам способы управления фазами ШД, поясняя ответ временными диаграммами? Какой способ управления используется в лабораторной работе?
2. Расскажите, поясняя ответ временной диаграммой, принцип работы D-триггера с динамическим управлением.
3. Объясните по схеме принцип работы распределителя импульсов.
4. Для чего предназначены диоды VD3-VD5 на схеме управления ШД (см. Рис. 16.5)?
5. В чем основное отличие в управлении ШД от контроллера и от электронной схемы?

# ЛАБОРАТОРНАЯ РАБОТА №17

## Управление светофорами регулируемого перекрестка

**Цель работы**

Разработка программы управления светофорами регулируемого перекрестка на основе заданного алгоритма в пакете GX Developer.

**Краткие теоретические сведения**

Для написания управляющей программы следует изучить следующие разделы 1 части данного издания:

11.1.6 Команды SET(Установить)/RST(Сбросить);

11.2 Программирование внутреннего реле (меркер);

11.3 Программирование счетчика;

11.4 Программирование таймера;

11.5 Программирование одиночных импульсов. Команды (PLF) и (PLS);

Для программирования мерцания, которое применяется при организации работы светофоров в ночном режиме и при переключении светофоров, целесообразно использовать вспомогательное реле M8013 (постоянное мерцание с интервалом в 1 секунду).

Пример:

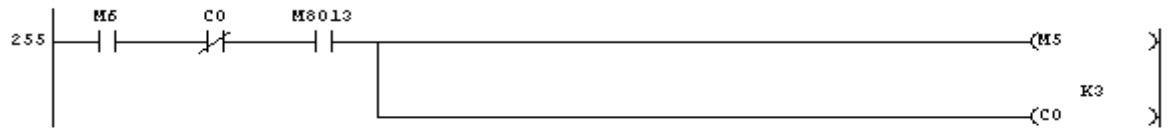



Рис. 17.1 – Пример реализации мерцания

Как только M6 переходит в ON, меркер M5 создает мерцание на выходе 3 раза с интервалом в 1 секунду и выход отключается. Обнулив счетчик с помощью команды RST, можно заново организовать мерцание.

Перекресток, на котором организуется движение транспортных средств с помощью светофоров, показан на рисунке 17.2.


Рис. 17.2 – Перекресток

Алгоритм движения на перекрестке определяется правилами дорожного движения.

При описании алгоритма работы и принципиальной схемы системы управления движением используются следующие обозначения:

- A,B,C,D,E – светофоры (смотри рисунки 17.2-17.4);
- Переключатель SW1(T3) (Пуск/Останов программы);
- Переключатель SW2(T2) (Специальный режим);
- Переключатель SW3(T1) (Дневной/ночной режим);
- Кнопка PB1 и PB2 (Включается разрешающий сигнал светофора для пешеходов);
- Кнопка PB3 (Имитация сигнала датчика подсчета въезжающих в паркинг автомобилей);
- Кнопка PB4 (Имитация сигнала датчика подсчета выезжающих из паркинга автомобилей);
- Кнопка PB5 (Резервная);
- X0-X7 – входы ПЛК;
- Внешний источник питания ($U_п$= +24 В, I=0,6 А);
- $Y_0$,$Y_1$,$Y_2$,$Y_3$ – выходные реле ПЛК для подачи сигналов красной(1), желтой(2), зеленой(3), зеленой(4) на лампы светофора А (смотри Рис. 17.4) и т.д.

Схемы подключения входов/выходов ПЛК приведены на рисунках 17.3, 17.4. Более подробно о подключении входов/выходов контроллера Mitsubishi FX0S 30MR-DC,который используется в лабораторной работе, изложено в 1 части данного издания:

3.4 Подключение источника питания;
3.5 Подключение входов;
3.6 Подключение выходов.

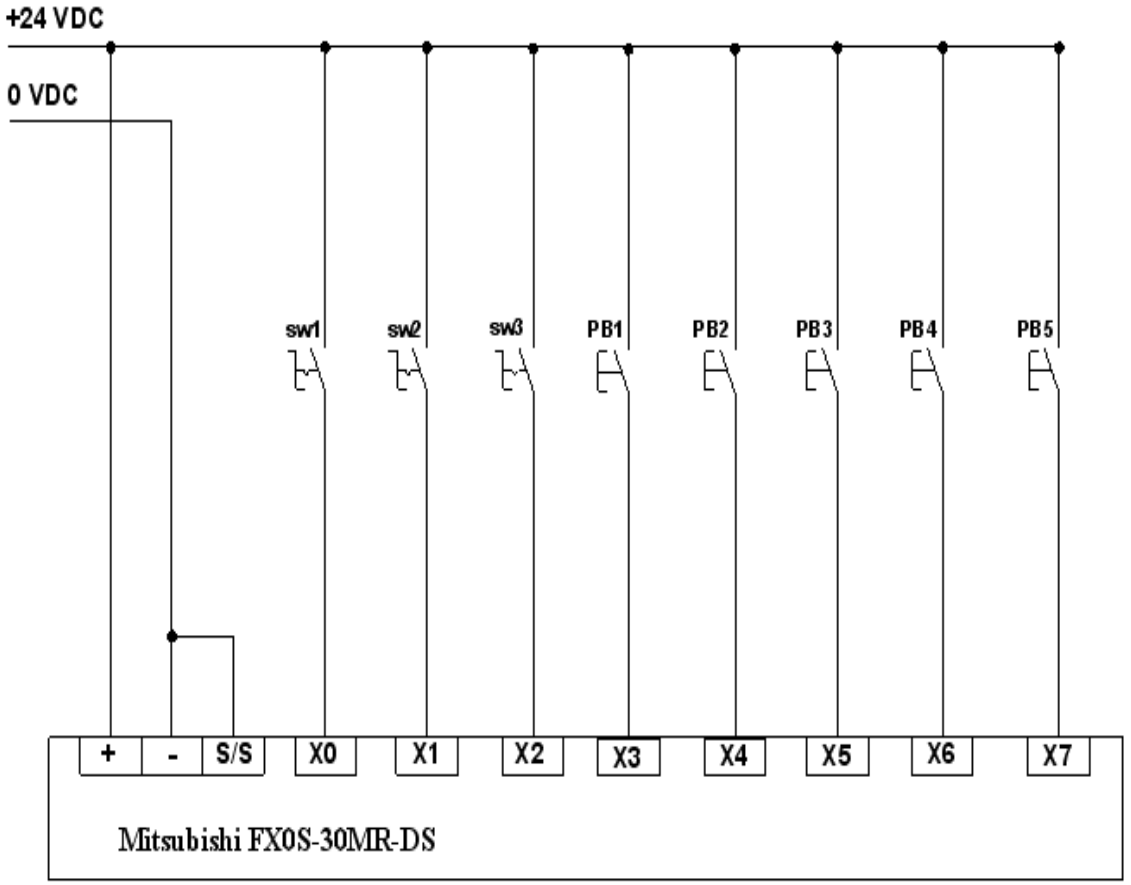



Рис. 17.3. Схема подключения входов ПЛК

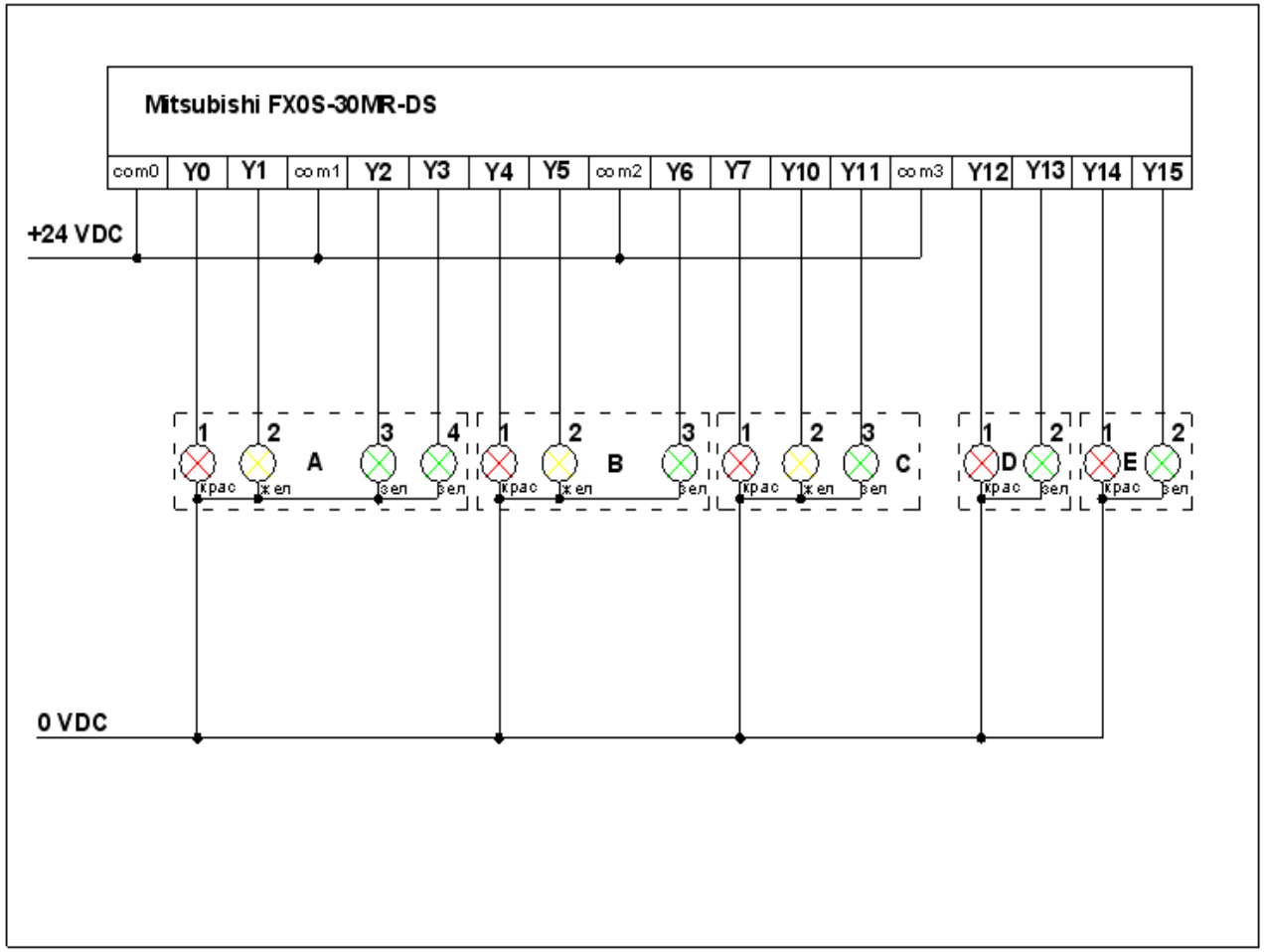



Рис. 17.4. Схема подключения выходов ПЛК

**Алгоритм управления светофорами**

1) При включении схемы переключателем S1 включаются светофоры A3,B3,C1,D1,E2 (смотри Рис. 17.2), при этом разрешается движение с Запада на Восток и с Востока на Запад, по главной дороге поворот на паркинг со стороны Востока, если при въезде в паркинг горит зеленый (E2); Разрешен выезд с паркинга на Запад, при отсутствии движущейся машины с левой стороны.

2) При включении желтого сигнала светофоров A2,B2,C2, гаснут A3, B3, продолжают гореть D1, C1, E2, нужно закончить движение транспортным средствам в различных направлениях, находящимся в пределах перекрестка, а приближающимся к нему остановиться у светофора, или, кому горел красный, (пешеходам или на выезде с паркинга) подготовиться к движению.

3) При включении красного сигнала светофора (загораются A4,A1,B1,C3; гаснут A2,B2,C2,C1; продолжают гореть D1,E2) на главной дороге (Восток – Запад) запрещается движение с Востока на Запад и наоборот, а разрешается выезд с паркинга на Восток, выезд с паркига на Запад, заезд на парковку с Запада, при наличии зеленого сигнала светофора при въезде в паркинг, предварительно пропустив все транспортные средства, которые могут создать помеху движения, т. е выезжающим с паркинга.

4) Загораются светофоры A2,B2,C2; гаснут: A4, C3; горят A1, B1, E2, D1

5) Разрешен выезд с паркинга на Запад, при отсутствии движущейся машины с левой стороны (загораются светофоры: A3,B3,C1; гаснут A1,A2,B1,B2,C2; горят: D1, E2;), то есть цикл повторяется.

6) Переключатель SW2 включает специальный режим – все светофоры включают запрещающий сигнал. По выключению этого переключателя цикл работы светофора должен начаться с начала (пункт 1).

7) Предусмотреть переход светофоров в режим работы в ночное время (переключатель SW3) – светофоры переходят в режим мигания желтого сигнала. По выключению этого переключателя цикл работы светофора должен начаться с начала (пункт 1).

8) Пешеходы кнопкой PB1 или PB2, включают разрешающий сигнал светофоров D2. Разрешающий сигнал светофора D2 загорается после нажатия кнопки PB1 или PB2 когда включаются лампы A1 и B1.

**Внимание**. При организации программы предусмотреть ограничение включения пешеходного светофора не более одного раза за полный цикл работы перекрестка.

9) Нажатием кнопки PB3 имитируется сигнал датчика контроля въезжающих в паркинг автомобилей. Одно нажатие кнопки PB3 – плюс 1 автомобиль в паркинге. Общее число мест парковки – 10. При отсутствии свободных мест, включается красный сигнал светофора E1, запрещающий въезд на парковку. Нажатием кнопки PB4 имитируется выезд автомобилей из паркинга - счетчик количества занятых мест обнуляется.

***Примечание***: перед переключением светофоров A,B,C с зеленого сигнала на желтый обеспечить мерцание зеленого 3 раза с интервалом в 1 секунду.

Цикл работы светофоров A,B,C:
красный горит - 30 сек;
красный/желтый(красный и желтый горят вместе) – 3 сек;
зеленый горит – 30 сек;
желтый горит – 3 сек;

Цикл работы светофора D:
зеленый горит по команде пешехода – 13 сек (кнопки PB1,PB2);

**Задание:**

1. Согласно алгоритму управления светофорами, составить программу для управления светофорами регулируемого перекрестка.
2. Произвести тестирование работы управляющей программы в режиме off-line посредством программы Ladder Logic Test (смотри лабораторную работу 15).
3. Убедившись в выполнении программой заданных условий, записать ее в память контроллера (смотри лабораторную работу 15) и продемонстрировать правильную работу на стенде.
4. Проверить работу системы управления в следующем порядке:
   - Включить источник питания;
     - Переключателем T3 запустить систему управления светофорами и убедиться в правильной работе светофоров в этом режиме;
     - Переключателем T2 включить специальный режим и убедиться в правильной работе светофоров в этом режиме;
     - Отключить специальный режим и удостовериться, что светофоры начинают работать с начала цикла, то есть с пункта 1 алгоритма управления;
     - переключателем T1 включить ночной режим и убедиться в правильной работе светофоров в этом режиме;
     - Отключить ночной режим и удостовериться, что светофоры начинают работать с начала цикла, то есть с пункта 1 алгоритма управления;
     - Выполнить пункт 8 алгоритма управления;
     - Выполнить пункт 9 алгоритма управления;
   - Отключить источник питания;

**Контрольные вопросы**

1. Для чего используются внутренние реле ПЛК?
2. В каких случаях используются инструкции PLS и PLF?
3. Какие способы организации мерцания вы знаете?
4. Назначение и виды счетчиков.
5. Объясните работу установки по принципиальной схеме.
6. Рассчитайте выходную нагрузку контроллера с учетом того что, потребляемый ток лампочки равен 50 мА.
7. Поясните, как организована защита контроллера при перегрузках.

# 2. ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРОГРАММИРОВАНИЮ МИКРОКОНТРОЛЛЕРОВ ATMega

## 2.1 Общие сведения о микроконтроллерах

Современную микроэлектронику трудно представить без такой важной составляющей, как микроконтроллеры (в дальнейшем МК). Микроконтроллерные технологии очень эффективны. Одно и то же устройство, которое раньше собиралось на традиционных элементах, будучи собрано с применением микроконтроллеров, становится проще. Оно не требует регулировки и меньше по размерам. Кроме того, с применением микроконтроллеров появляются практически безграничные возможности по добавлению новых потребительских функций и возможностей к уже существующим устройствам. Достаточно просто поменять программу.

ATMega8 — 8-разрядный КМОП микроконтроллер, основанный на архитектуре Atmel AVR. Контроллер выполняет большинство инструкций за 1 такт, поэтому вычислительная мощность контроллера равна 1MIPS на 1 Мгц.

МК имеет RISC-архитектуру, но формат команды двухоперандный, за один такт может быть обращение только к двум регистрам. Контроллер содержит 32 регистра, которые могут равноправно использоваться в арифметических операциях.

Основные аппаратные характеристики МК:

* 8 Кбтфлеш-памяти команд;
* 512 байт электрически программируемой памяти;
* 1 Кбайт статической памяти;
* 23 линии ввода/вывода общего назначения;
* 32 РОНа;
* три многоцелевых таймер-счётчика с режимом сравнения;
* поддержка внутренних и внешних прерываний;
* универсальный асинхронный адаптер;
*байт-ориентированный двухпроводной последовательный интерфейс;
* 6/8 канальный АЦП с точностью 8 и 10 двоичных разрядов;
* сторожевой таймер;
* последовательный порт SPI;
* расширенные режимы управления энергопотреблением.

Ядро микроконтроллера - AVR.

Количество записей во флэш-память – 10 000 раз.

Рассмотрим, что же он из себя внешне представляет (рис.2.1).

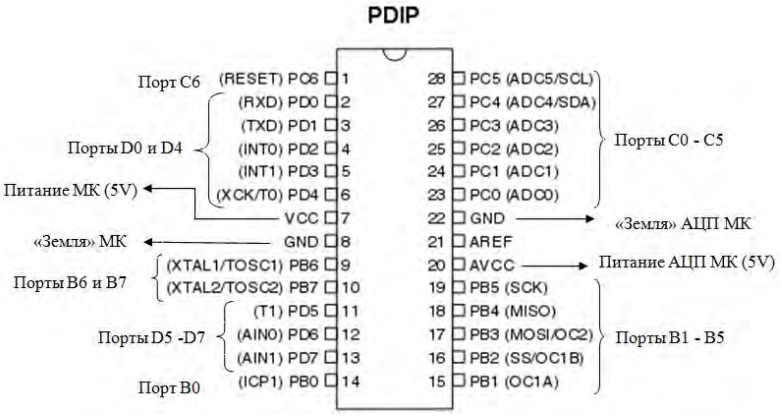



Рис. 2.1. МК Atmega8 в корпусе DIP

На рис. 1.1 приведен МК Atmega8 в DIP корпусе (существуют и другие корпуса) с обозначением ножек. Кратко изучим их назначение (подробно изучим с лабораторных работах):

1. все то, что имеет обозначение PX – это порты где, X – номер порта. На все порты можно подавать сигналы или принимать, что и будем делать в лабораторных работах;
2. питание МК в 5 вольт подается на ножки VCC и GND;
3. программирование МК производится по ножкам: 1 (reset), 17 (MOSI), 18 (MISO), 19 (SCK)с помощью программатора.

Специально разработанные лабораторные работы построены таким образом, чтобы любой человек смог изучить язык программирования (в данном случае C++, но существует и другие языки) от практически нулевого уровня, до уровня, позволяющего писать программы средней сложности.

Каждая новая лабораторная работа начинается с постановки задачи. Затем вы можете увидеть процесс построения алгоритма и, наконец, увидите, как создается управляющая программа для наших задач.

Все программные работы и примеры к ним приведены на языке СИ, который поддерживается средой программирования МК AVR – CodeVisionAVR.

## 2.2 **Введение в язык** C++ **и** CodeVisionAVR

CodeVisionAVR — это кросс-компилятор СИ, интегрированная среда разработки (IDE — Integrated Development Environment) и автоматический генератор программ (CodeWizardAVR), разработанные для семейства AVR-МК фирмы Atmel.

Программа является 32-битовым приложением, которое работает под операционными системами Windows 95, 98, NT 4, 2000 , XP, 7.

CodeVisionAVR обеспечивает выполнение почти всех элементов языка C++ ( в дальнейшем просто «С» или «Си»), которые разрешены архитектурой AVR, с некоторыми добавленными характеристиками, которые реализуют преимущество специфики архитектуры AVR.

Программное обеспечение CodeVision может работать с такими программаторами серии Atmel, как с STK500/AVRISP/AVRProg , KandaSystems STK200+/300, Dontronics DT006, VogelElektronik VTEC-ISP, Futurlec JRAVR и платой разработчика MicroTronics ATCPU/Mega2000.

Кроме стандартных библиотек СИ, компилятор СИ CodeVisionAVR имеет библиотеки для:

- алфавитно-цифровых LCD-модулей;
- шины I2C от Philips;
- температурного датчика LM75 от National Semiconductor;
- часов реального времени PCF8563, PCF8583 от Philips и DS1302, DS1307 от Dallas Semiconductor;
- протокола 1 -Wire от Dallas Semiconductor;
- температурного датчика DS1820/DS18S20 от Dallas Semiconductor;
- термометра/термостата DS1621 от Dallas Semiconductor;
- EEPROM DS2430 и DS2433 от Dallas Semiconductor;
- SPI;
- управления питанием;
- задержек;
- преобразования кода Грея.

CodeVisionAVR также содержит автоматический генератор программ - CodeWizardAVR, который позволяет написать за несколько минут весь код, необходимый для выполнения следующих функций:

- установка доступа к внешней памяти;
- идентификация источника сброса чипа;
- инициализация порта ввода/вывода;
- инициализация внешних прерываний;
- инициализация таймеров/счётчиков;
- инициализация сторожевого таймера;
- инициализация UART и прерываний, управляющих буфером последовательной связи;
- инициализация аналогового компаратора;
- инициализация АЦП;
- инициализация интерфейса SPI;

- инициализация шины 12C, температурного датчика LM75, термометра/термостата DS1621 и часов реального времени PCF8563, PCF8583, DS1302, DS1307;
- инициализация шины 1-Wire и температурного датчика DS1820/DS18S20;
- инициализация LCD-модуля.

В лабораторных работах приводятся лишь начальные сведения о языке СИ и, где необходимо, поясняются специфические особенности реализации языка компилятором СИ CodeVisionAVR.

Препроцессор (макропроцессор) — это составная часть языка СИ, которая обрабатывает исходный текст программы до того, как он пройдет через компилятор. Препроцессор читает строки текста и выполняет действия, определяемые командными строками. Если первым символом в строке, отличным от пробела, является символ #, то такая строка рассматривается препроцессором как командная. Командные строки называются директивами препроцессора.

Директивы препроцессора позволяют:
- включать в программу текст из других файлов;
- передавать компилятору специальные директивы;
- определять макросы, которые облегчают программирование и улучшают удобочитаемость исходного кода;
- задавать условия компиляции для отладочных целей и/или для уменьшения размера получаемого кода.

Препроцессор компилятора CodeVisionAVR имеет несколько директив. В табл. 2.1 даётся их краткое описание.

Таблица 2.1.

Директивы препроцессора компилятора CodeVisionAVR

| Директива | Назначение |
|---|---|
| #include | Используется для включения в программу другого файла |
| #define | Используется для замены одних лексических единиц языка Си на другие, а также для генерации макросов |
| #undef | Используется для отмены действия директивы #define |
| #if | Используются для условной компиляции |
| #ifdef | |
| #ifndef | |
| #else | |
| #endif | |

| | |
|---|---|
| #line | Используется для изменения встроенных макросов _LINE_ и _FILE_ |
| #error | Позволяет остановить компиляцию и отобразить сообщение об ошибке |
| #asm | Используются для включения в программу ассемблерного кода |
| #endasm | |
| #pragma… | Разрешает специальные директивы компилятора |

В лабораторных работах будет использовать только директива include, которая позволяет в дальнейшем использовать функции и команды, не разрешенные в СИ.

Операнд — это константа, литерал, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное комбинацией операндов, знаков операций и круглых скобок. Иными словами, операнд – это математическая функция. Каждый операнд имеет тип (табл. 2.2).

В языке СИ переменные делятся на типы. Переменная каждого типа может принимать значения из одного определенного диапазона. Например:

- переменная типа char — это только целые числа;
- переменная типа float — вещественные числа (десятичная дробь) и т. д.

Использование переменных нескольких фиксированных типов — это отличительная особенность любого языка высокого уровня. Разные версии языка СИ поддерживают различное количество типов переменных. Версия СИ, используемая в CodeVisionAVR, поддерживает тринадцать типов переменных (табл. 2.3).

Таблица 2.3.

Типы переменных языка СИ в CodeVisionAVR

| Название | Количество бит | Значение |
|---|---|---|
| bit | 1 | 0 или 1 |
| char | 8 | -128-127 |
| unsigned char | 8 | 0-255 |
| signed char | 8 | -128-127 |
| int | 16 | -32768-32767 |
| short int | 16 | -32768-32767 |
| unsigned int | 16 | 0-65535 |
| signed int | 16 | -32768-32767 |
| long int | 32 | -2147483648 – 2147483647 |
| unsigned long int | 32 | 0-4294967295 |

| | | |
|---|---|---|
| signed long int | 32 | -2147483648 - 2147483647 |
| float | 32 | ±1.175e-38- ±3.402e38 |
| double | 32 | ±1.175e-38 — ±3.402e38 |

Таблица 2.2.

Операнды

| Уровень | Операторы | Категория | Описание |
|---|---|---|---|
| 1 | () | | Круглые скобки |
| | [] | | Элемент массива |
| | . | Доступ к данным | Обращение к элементу структуры, например: `PORTB.1` — разряд 1 порта B |
| | -> | | Обращение к элементу структуры, определенной указателем, например: `pStruct->x` — элемент `x` структуры, на которую указывает `pStruct` |
| | ++, -- (постфиксы) | Арифметические | Операторы автоинкремента и автодекремента после того как выражение, в котором задействованы соответствующие операнды, вычислено. Примеры: `a = b++;` равнозначно `a = b; b = b+1;` `a = b--;` равнозначно `a = b; b = b-1;` |
| 2 | ++, -- (префиксы) | | Операторы автоинкремента и автодекремента перед тем как выражение, в котором задействованы соответствующие операнды, будет вычислено. Примеры: `a = ++b;` равнозначно `b = b+1; a = b;` `a = --b;` равнозначно `b = b-1; a = b;` |
| | ! | Логические | Логическое (унарное) отрицание |
| | ~ | Поразрядные | Поразрядное отрицание |
| | & | Доступ к данным | Адрес |
| 3 | +, - (унарные) | Арифметические | Изменение знака операнда |
| | * (унарный) | Доступ к данным | Разыменование указателя |
| 4 | * (бинарный) | Арифметические | Умножение |
| | / | | Деление |
| | % | | Остаток от деления |
| 5 | +, - (бинарные) | | Сложение и вычитание |
| 6 | << | Поразрядные | Поразрядный сдвиг влево |
| | >> | | Поразрядный сдвиг вправо |
| 7 | <, >, <=, >= | Сравнения | Меньше, больше и т.д. |
| 8 | ==, != | | Равно, не равно |
| 9 | & | Поразрядные | Поразрядное "И" |
| 10 | ^ | Поразрядные | Поразрядное "Исключающее ИЛИ" |
| 11 | \| | Поразрядные | Поразрядное "ИЛИ" |
| 12 | && | Логические | Логическое "И" |
| 13 | \|\| | Логические | Логическое "ИЛИ" |
| 14 | = | Присваивание | Возможны также сочетания оператора присваивания с арифметическими и поразрядными операторами, например: `a += b;` равнозначно `a = a + b;` `a *= b+c;` равнозначно `a = a * (b + c);` |

В языке СИ любая переменная, прежде чем будет использована, должна быть описана. При описании задается ее тип. В дальнейшем диапазон принимаемых значений должен строго соответствовать выбранному типу переменной. Описание переменной и задание типа необходимы потому, что оттранслированная с языка СИ программа выделяет для хранения значений каждой переменной определенные ресурсы памяти.

Это могут быть ячейки ОЗУ, регистры общего назначения или даже ячейки EEPROM или Flash-памяти (памяти программ). В зависимости от заданного типа, выделяется различное количество ячеек для каждой конкретной переменной. Описывая переменную, мы сообщаем транслятору, сколько ячеек выделять и как затем интерпретировать их содержимое. Посмотрим,

как выглядит строка описания переменной в программе. Она представляет собой запись следующего вида:

Тип Имя;

где «Тип» — это тип переменной, а «Имя» — ее имя.

Имя переменной выбирает программист. Допускается использование только латинских букв, цифр и символа подчеркивания. Начинаться имя должно с буквы или символа подчеркивания.

Кроме арифметических и логических выражений язык СИ использует функции. В отличие от математических функций, функции языка СИ не всегда имеют входные значения и даже не обязательно возвращают результат. Далее на конкретных примерах мы увидим, как и почему это происходит.

Вообще, роль функций в языке СИ огромная. Программа на языке СИ просто-напросто состоит из одной или нескольких функций. Каждая функция имеет свое имя и описание. По имени производится обращение к функции. Описание определяет выполняемые функцией действия и преобразования. Вот как выглядит описание функции в программе СИ:

тип Name (список параметров) {

тело функции }

Здесь Name — это имя функции. Имя для функции выбирается по тем же правилам, что и для переменной. При описании функции перед ее именем положено указать тип возвращаемого значения. Это необходимо транслятору, так как для возвращаемого значения он тоже резервирует ячейки.

Если перед именем функции вместо типа возвращаемого значения записать слово void, то это будет означать, что данная функция не возвращает никаких значений. В круглых скобках после имени функции записывается список передаваемых в нее параметров.

Функция может иметь любое количество параметров. Если параметров два и более, то они записываются через запятую. Перед именем каждого параметра также должен быть указан его тип. Если у функции нет параметров, то в скобках вместо списка параметров должно стоять слово void. В фигурных скобках размещается тело функции:

```
void main (void) {a=5;}
```

В этом случае операторы, составляющие тело функции, разделяет только точка с запятой. Вот пример такой записи:

```
тип Name (список параметров) {тело функции }

void main(void) {a=5;c=6;}
```

Любая программа на языке СИ должна обязательно содержать одну главную функцию. Главная функция должна иметь имя main. Выполнение программы всегда начинается с выполнения функции main. Функция main в данной версии языка СИ никогда не имеет параметров и никогда не возвращает никакого значения.

Тело функции, кроме команд, может содержать описание переменных. Все переменные должны быть описаны в самом начале функции, до первого оператора. Такие переменные могут быть использованы только в той функции, вначале которой они описаны. Вне этой функции данной переменной как бы не существует.

Если вы объявите переменную в одной функции, а примените ее в другой, то транслятор выдаст сообщение об ошибке. Это дает возможность объявлять внутри разных функций переменные с одинаковыми именами и использовать их независимо друг от друга.

Глобальная переменная объявляется не внутри функций, а в начале программы, еще до описания самой первой функции. Не спешите без необходимости делать переменную глобальной. Если программа достаточно большая, то можно случайно присвоить двум разным переменным одно и то же имя, что приведет к ошибке. Такую ошибку очень трудно найти.

Начнем изучение СИ с описания нам пока неизвестных используемых там команд, которые пригодятся в при выполнении лабораторных работ.

include

Оператор присоединения внешних файлов. В 1 строке нашей будущей программы этот оператор присоединяет к основному тексту программы стандартный текст описаний для микроконтроллера Atmega8.

while

Оператор цикла. Форма написания команды whi1e очень похожа на форму описания функции. В общем случае команда while выглядит следующим образом:

while (условие){тело цикла}

Перевод английского слова whi1e — «пока». Эта команда организует цикл, многократно повторяя тело цикла до тех пор, пока выполняется «условие», то есть пока выражение в скобках является истинным. В языке СИ принято считать, что выражение истинно, если оно не равно нулю, и ложно, если равно.

Комментарии

В программе на языке СИ широко используются комментарии, которые не позволят вам запутаться при написании программы. Принято два способа написания комментариев.

Первый способ — использование специальных обозначений начала и конца комментария. Начало комментария помечается парой символов /*, а конец комментария символами */. Это выглядит следующим образом:

```
/* Комментарий */
```

Причем комментарий, выделенный таким образом, может занимать не одну, а несколько строк.

Второй способ написания комментария — двойная наклонная черта (//). В этом случае комментарий начинается сразу после двойной наклонной черты и заканчивается в конце текущей строки.

В дальнейшем наши программы будут начинаться с заголовка, в нашем случае это многострочный комментарий, который мастер поместил с информацией о том, что программа создана при помощи CodeWizardAVR. Также автоматически будут указаны такие параметры, как тип процессора, его тактовая частота частоту, модель памяти (mega — означает большая модель), размер используемой внешней памяти и размер стека. После комментариев будет начинаться вся основная часть программы, которую мы и будем учиться писать.

## 2.3 Моделирование схем с помощью программы Proteus

### 2.3.1 Описание

Proteus Professional представляет собой систему схемотехнического моделирования, базирующуюся на основе моделей электронных компонентов принятых в PSpice. Отличительной чертой пакета Proteus Professional является возможность моделирования работы программируемых устройств: микроконтроллеров, микропроцессоров, DSP и прочее. Дополнительно в пакет Proteus Professional входит система проектирования печатных плат. Proteus Professional может симулировать работу следующих микроконтроллеров: 8051, ARM7, AVR, Motorola, PIC, BasicStamp. Библиотека компонентов содержит справочные данные.

Поддерживает МК: PIC, 8051, AVR, HC11, ARM7/LPC2000 и другие распространенные процессоры. Более 6000 аналоговых и цифровых моделей устройств. Работает с большинством компилятором и ассемблерами.

PROTEUS VSM позволяет очень достоверно моделировать и отлаживать достаточно сложные устройства, в которых может содержаться несколько МК одновременно, и даже разные семейства в одном устройстве. Нужно только ясно понимать, что моделирование электронной схемы не абсолютно точно повторяет работу реального устройства. Вместе с тем  для отладки алгоритма работы МК, этого более чем достаточно. PROTEUS содержит огромную библиотеку электронных компонентов. Отсутствующие модели можно дополнить.

Если компонент не программируемый, то нужно на сайте производителя скачать его SPICE модель и добавить в подходящий корпус.

Proteus 7 состоит из двух основных модулей:

ISIS - графический редактор принципиальных схем служит для ввода разработанных проектов с последующей имитацией и передачей для разработки печатных плат в ARES. К тому же после отладки устройства можно сразу развести печатную плату в ARES, которая поддерживает авторазмещение и трассировку по уже существующей схеме.

ARES - графический редактор печатных плат со встроенным менеджером библиотек и автотрассировщиком ELECTRA, автоматической расстановкой компонентов на печатной плате.

PROTEUS имеет уникальные возможности.

USBCONN - этот инструмент позволяет подключиться к реальному USB порту компьютера.

COMPIM - этот компонент позволяет вашему виртуальному устройству подключиться к реальному COM-порту вашего ПК.

Примеры:

- вы можете подключить через "шнурок" к свободному COM-порту сотовый телефон и отлаживать устройство на МК, которое должно управлять им.

- вы можете подключить к COM-порту любое реальное устройство с которым ваш создаваемый прибор будет общаться в реальности!

PROTEUS VSM - великолепно работает с популярными компиляторами Си для МК:

- CodeVisionAVR (для МК AVR)
- IAR (для любых МК)
- ICC (дляМК AVR, msp430, ARM7, Motorola)
- WinAVR (для МК AVR)
- Keil (для МК архитектуры 8051 и ARM)
- HiTECH (для МК архитектуры 8051 и PIC от Microchip)

Для выполнения лабораторных работ необходимо смоделировать в Proteus лабораторный стенд. В данной работе была уже показана приблизительная модель стенда на рисунке 3.1. Теперь мы остановимся на этом месте, чтобы подробнее рассмотреть создание в Proteus схемы и её моделирования.

### 2.3.2 Работа с Proteus

Начнем с открытия программы, установку программы попросите у преподавателя. Нам нужно открыть программу с названием ISIS, путь к ней лежит через C:\Program Files\LabcenterElectronics\Proteus 7 Professional\BIN. После запуска ISIS.exe нам откроется рабочее окно, в котором мы собственно и будем создавать нашу схему (рис.2.1)

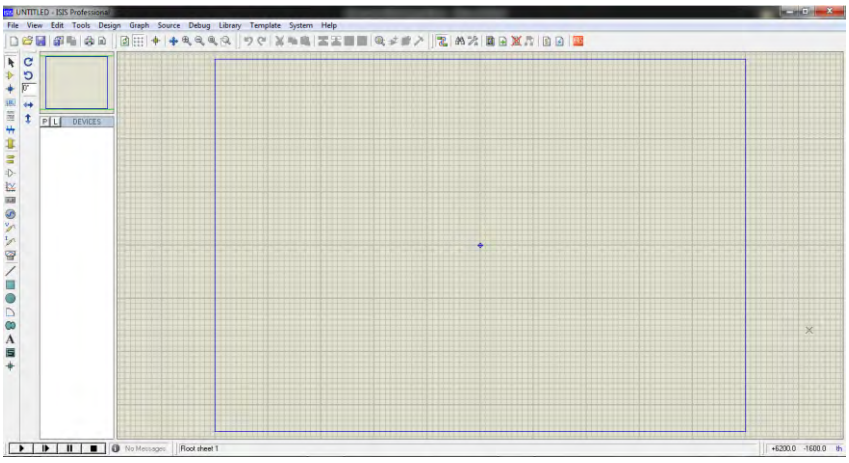


Рис. 2.1. Рабочее окно программы Proteus

Теперь нам необходимо попасть в библиотеку программы, в которой уже имеются все нужные элементы. В этой библиотеке собранны все возможные элементы, используемые в создании принципиальных схем. Чтобы вызвать меню библиотеки нужно найти кнопку слева Pick from Libraries(рис.2.2).

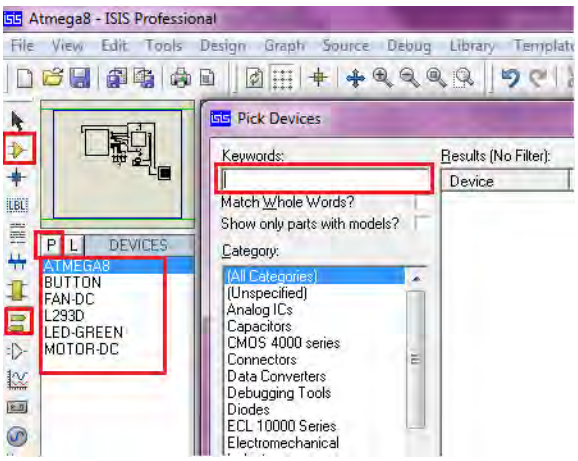



Рис. 2.2. Основные кнопки для создания схем:
1 -Pick from Libraries; 2 -Devices; 3 -Terminals Mode

В открывшемся окне есть строка Keywords, здесь мы вводим слово или часть слова и при нажатии Enter получаем все имеющиеся элементы в базе по введенному запросу. Так же можно найти необходимые элементы с помощью уже имеющихся тут категорий (Category). Справа мы можем увидеть схематический вид элемента, который будет применяется на схеме и натуральный вид со всеми размерами (этого вида может не быть). После выбора элемента кнопкой Ok или двойным щелчком, элемент условно помещается на конце вашего курсора-карандаша. Чтобы поместить элемент в нужную вам область, наведите туда курсор и кликните 1 раз. Так же программа сохраняет все выбранные элементы в данном проекте в специальном поле, с помощью которого можно миновав библиотеку вставлять нужные нам элементы (рис.3.2). Сверху также отображаются выбранные вами элементы. Для нашего стенда необходимы следующие:

1) ATMEGA 8. микроконтроллер
2) LED-GREEN. светодиоды для индикациисзеленым свечением
3) BUTTON. кнопка
4) MOTOR-DC.мотор с индикацией количества оборотов
5) L293D. драйвер

Так же нам нужно подключить питание и «землю» к схеме. Для этого найдем объект с названием Terminals Mode(рисунок 3.2).В открытом поле находим Groundи Power. Устанавливаем их к выводам микросхем и подсоединяем «карандашом» соответствующие им контакты.

Все объекты в Proteus либо интеллектуально подсоединены на питание (если соответствующие выводы отсутствуют как, например, у МК ATMega8), либо нужно подсоединить (при наличии выводов у данного объекта). При этом достаточно просто соединить вывод Vcc электронного компонента с объектом Power(означает питание), а GNDс объектом Ground. Значения напряжения и тока для объекта выставляются автоматически при подключении выводов питания названных выше!

Далее все элементы на схеме соединяем проводами (в данном случае линиями). Для этого наводим на элемент курсор-карандаш, ищем места,

специально отведенные для линий. В перекрестии курсора-карандаша появится маленький треугольник, зажимаем левую клавишу мыши и тянем линию к другому элементу, находим такой же место с маленьким треугольником и отпускаем клавишу. Аналогично соединяем все элементы.

Чтобы условно записать программу в виртуальный контроллер, в программе Proteus нужно дважды кликнуть по контроллеру. В открывшемся окне нужно указать путь к созданному вами hex-файлу в строке Program File, после нажать кнопку Ok (рис.2.3).

Кроме того, в окне настройки виртуального МК есть поля, где следует установить параметры согласно рис. 2.3.

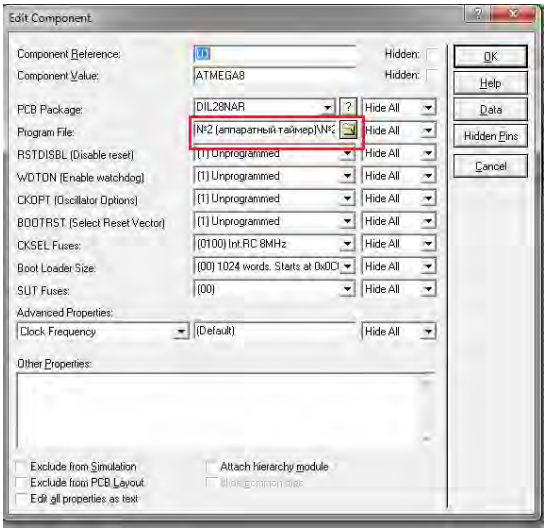
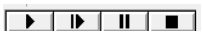



Рис.2.3.Запись hex файла в МК

Управление смоделированной схемой производится с помощью кнопок  .

Если все сделано правильно, то при запуске модели вы увидите точно такую же работу МК, как и на реальном стенде.

Для корректного отображения скорости вращения двигателя в Proteus,необходимо выбрать соответствующие параметры используемого двигателя (рис.2.4).

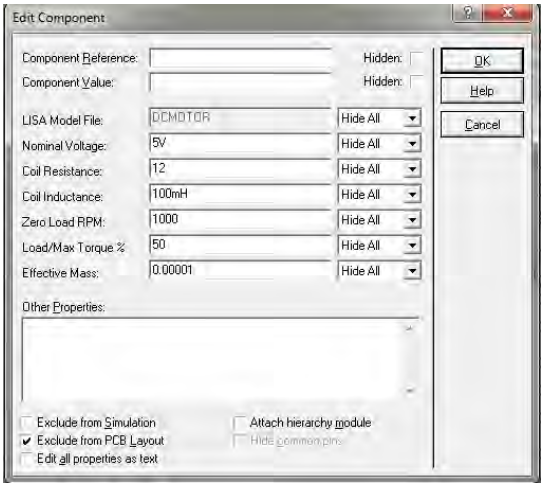



Рис. 2.4. Изменение параметров двигателя:

nominal voltage – номинальное напряжение; coil resistance – сопротивление катушки; coil inductance – катушки индуктивности; Zero load – нулевая нагрузка; Torque - крутящий момент; Effective mass - эффективная масса. Выставляем все эти значения согласно рис.2.4.

Для исследования изменяемых характеристик в Proteusможно использовать осциллограф (особенно понадобиться в следующих лабораторных работах). Она находится в Virtual Instruments Mode(рис.2.5).

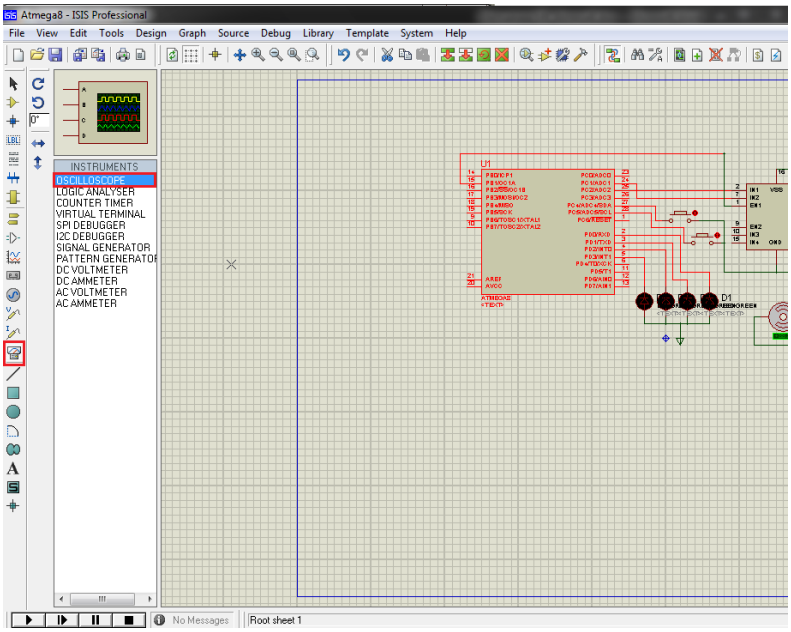



Рис. 2.5. Выбор осциллографа

Подключаем один из 4 каналов к нужной части схемы и получаем нужные нам характеристики. Обратите внимание на то, что осциллограф может выдавать преобразованные сигналы в зависимости от того, на какой вывод вы подключились (рисунок сигнала на картинке осциллографа в Proteusсоответствует получаемому сигналу). Вызов осциллографа происходит из вкладки Debug>Digital Oscilloscope в режиме симуляции.

## 2.4 Описание лабораторного стенда

Устройство МК ATMega8 мы изучили. Осталось разобраться в том, как правильно все подключить (рис. 2.6).

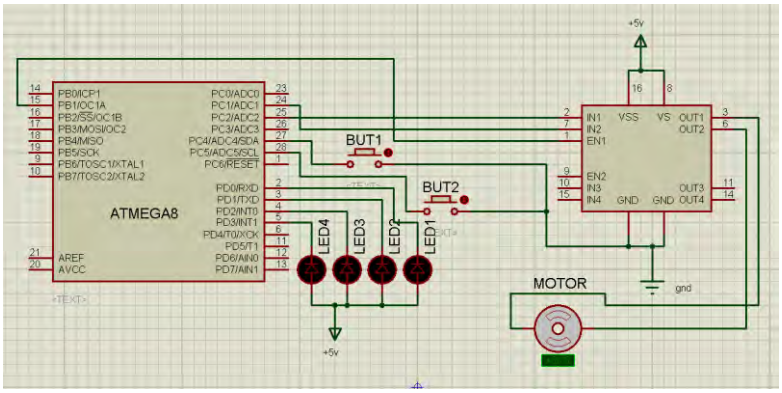



Рис. 2.6. Принципиальная схема лабораторного стенда

На рис.4.1 изображено подключение драйвера к двигателю, сам двигатель, 2 кнопки и 4 светодиода, подключенные к микроконтроллеру. Светодиоды подключаются к МК через резисторы 220 Ом, из-за свойств пропускания тока диодами. Как все это работает, будет зависеть от того, как мы напишем программу. Осталось разобраться в назначении драйвера для двигателя L293D.

Для управления двигателями необходимо устройство, которое бы преобразовывало управляющие сигналы малой мощности в токи, достаточные для управления моторами. Такое устройство называют **драйвером двигателей**.

L293D содержит сразу два драйвера для управления электродвигателями небольшой мощности (четыре независимых канала, объединенных в две пары). Имеет две пары входов для управляющих сигналов и две пары выходов для подключения электромоторов. Кроме того, у L293D есть два входа для включения каждого из драйверов. Эти входы используются для управления скоростью вращения электромоторов с помощью широтно-модулированного сигнала (ШИМ).

L293D обеспечивает разделение электропитания для микросхемы и для управляемых ею двигателей, что позволяет подключить электродвигатели с большим напряжением питания, чем у микросхемы. Разделение электропитания микросхем и электродвигателей может быть также необходимо для уменьшения помех, вызванных бросками напряжения, связанными с работой моторов.

Принцип работы каждого из драйверов, входящих в состав микросхемы, идентичен, поэтому рассмотрим принцип работы одного из них (рис. 2.7).

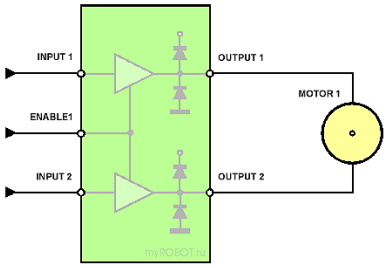



Рис. 2.7. Схема драйвера двигателей

К выходам OUTPUT1 и OUTPUT2 подключим электромотор MOTOR1.

На вход ENABLE1, включающий драйвер, подадим сигнал (соединим с положительным полюсом источника питания +5V). Если при этом на входы INPUT1 и INPUT2 не подаются сигналы, то мотор вращаться не будет.

Если вход INPUT1 соединить с положительным полюсом источника питания, а вход INPUT2 - с отрицательным, то мотор начнет вращаться.

Если соединить вход INPUT1 с отрицательным полюсом источника питания, а вход INPUT2 - с положительным. Мотор начнет вращаться в другую сторону.

Подадим сигналы одного уровня сразу на оба управляющих входа INPUT1 и INPUT2 (соединить оба входа с положительным полюсом источника питания или с отрицательным) - мотор вращаться не будет.

Если мы уберем сигнал с входа ENABLE1, то при любых вариантах наличия сигналов на входах INPUT1 и INPUT2 мотор вращаться не будет.

Представить лучше принцип работы драйвера двигателя можно, рассмотрев таблицу 2.4.

Таблица 2.4

Логика работы драйвера

| Enable | Input 1 | Input 2 | Output 1 | Output 2 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Теперь рассмотрим назначение выводов микросхемы L293D (рис. 2.8).

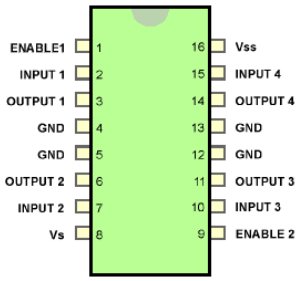



Рис. 2.8. Назначение выводов L293D

Входы ENABLE1 и ENABLE2 отвечают за включение каждого из драйверов, входящих в состав микросхемы.

Входы INPUT1 и INPUT2 управляют двигателем, подключенным к выходам OUTPUT1 и OUTPUT2.

Входы INPUT3 и INPUT4 управляют двигателем, подключенным к выходам OUTPUT3 и OUTPUT4.

Контакт Vs соединяют с положительным полюсом источника электропитания двигателей или просто с положительным полюсом питания, если питание схемы и двигателей единое. Проще говоря, этот контакт отвечает за питание электродвигателей.

Контакт Vss соединяют с положительным полюсом источника питания (+5V). Этот контакт обеспечивает питание самой микросхемы.

Четыре контакта GND соединяют с "землей" (общим проводом или отрицательным полюсом источника питания). Кроме того, с помощью этих контактов обычно обеспечивают теплоотвод от микросхемы, поэтому их лучше всего распаивать на достаточно широкую контактную площадку.

Характеристики микросхемы L293D:

1. напряжение питания двигателей (Vs) - 4,5...36V
2. напряжение питания микросхемы (Vss) - 5V
3. допустимый ток нагрузки - 600mA (на каждый канал)
4. пиковый (максимальный) ток на выходе - 1,2A (на каждый канал)
5. логический "0" входного напряжения - до 1,5V
6. логическая "1" входного напряжения - 2,3...7V
7. скорость переключений до 5 kHz
8. защита от перегрева.

На рис. 2.9 приведена схема простейшего LPT-программатора. Он состоит всего из 5 проводов, подключаемых к портам МК, поэтому он так и называется «Пять проводков».

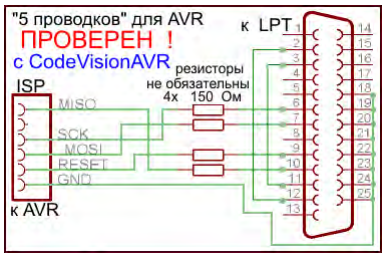



Рис. 2.9. Схема LPT-программатора

Резисторы в 150 Ом нужны для защиты LPT порта от тока, который выдает МК. Провода напрямую подключают к ножкам МК и можно программировать. Для надежности программирования ножки МК Resetи Vcc соединяют резистором в 10кОм, Resetи GND–керамическим конденсатором на 0,1 – 0,015 мкФ. После сборки такого программатора при наличии LPT порта можно запрограммировать любой микроконтроллер AVR, что мы и научимся делать в лабораторных работах.

Недостатком LPT программаторов является их длина проводов – не больше 15 см, а если больше, то запись либо медленная, либо с ошибками. Единственный способ использовать длинные провода – это чередовать каждый сигнальный провод с землей или использовать качественный экранированный кабель.

Очень важно знать! При программировании МК, сам КМ должен питаться от 5 Вольт.

Примечания программатору

После того, как программа написана, осталось записать ее в МК. Для этого выполняем действия согласно рис. 2.10, 2.11 (если программатор LPT) или согласно рис. 2.12 (если программатор USB).

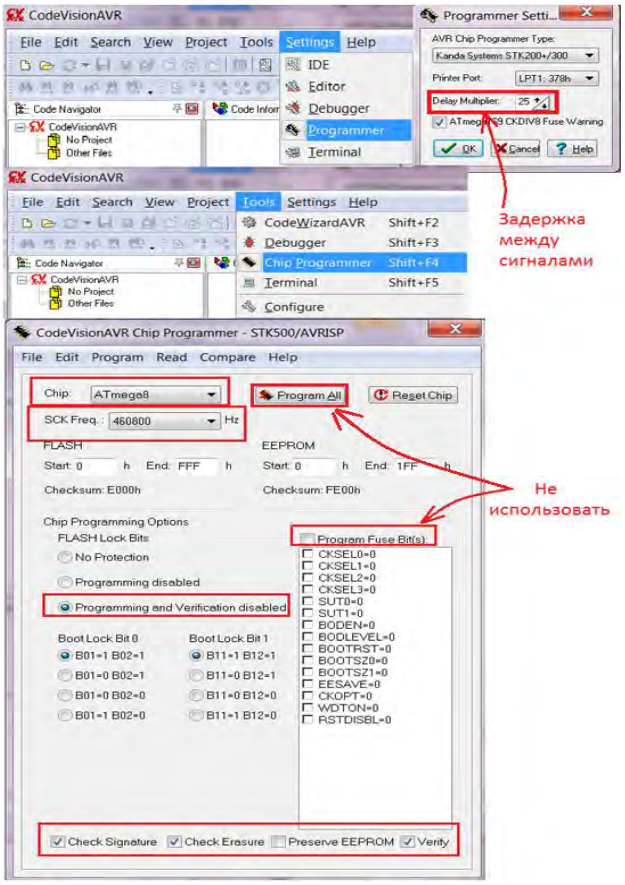



Рис. 2.10. Запись программы в МК

Программатор у нас серии STK 200+/300 (т.е. LPT). Если используется программатор USBAVR910, описание работы с ним смотрите ниже. Задержка между сигналами влияет на скорость записи и используется, если у программатора длинные провода. Используется порт LPT1. Настройку SCK Freq можно менять, она влияет на частоту строба, которая также может помочь, если у программатора длинные провода. Все остальные настройки выставить согласно рис. 2.10.

Часть рисунка подписана как «Не использовать» - это установка фьюзов и команда program all(запись всего, в том числе и фьюзов даже, если не стоит галочка Program Fuses).Фьюзы–особые настройки для изменения некоторой функциональности МК. Их перепрограммирование без особых знаний приведет к тому, что МК будет исправен, но вы ничего не сможете с ним сделать в силу, блокировки ножек отвечающих за последовательное программирование. Чтобы его восстановить, необходима дорогостоящая аппаратура (к примеру, параллельный программатор или реаниматор фьюзов)!!!!

После того, как настроили программатор согласно рис. 2.10, можно приступить к программированию (рис.2.11).

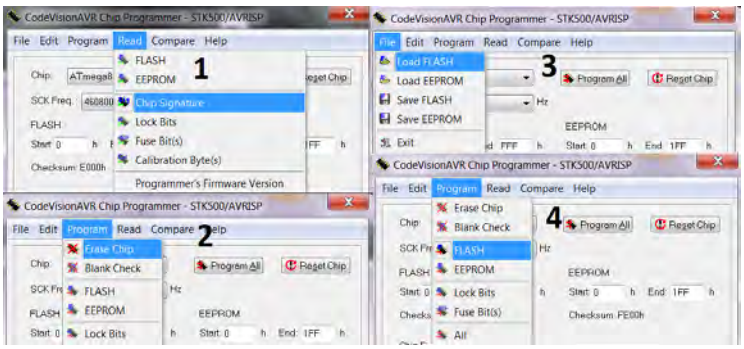



Рис. 2.11. Последовательность действий для записи программы в МК

Сначала проверяем сигнатуру чипа (действие 1). Если появилось окно с ошибкой, то стоит проверить подключение, в частности питание МК; попробуйте увеличить задержку в настройках программатора. Если ничего не выходит, то следует проверять всю схему целиком, а также работоспособность LPT порта у компьютера.

Если же получили окно, в котором программа увидела, что это ATmega8, то приступим к стиранию чипа (действие 2). Иногда появляется окно с ошибкой стирания – это может быть дефект из-за длины проводов программатора (следует опять же увеличить задержку) или сбой в самом МК (следует выбрать команду BlankCheck, которая исправит программный сбой).

После того, как стирание флэш-памяти чипа произошло успешно, можно записать в него новую программу. Для этого нужно загрузить ее в буфер обмена (действие 3), после команды Load FLASH в появившемся окне выбираем файл программы (имеет расширение hex в папке вашей программы) – встречали его при компиляции – это HEX-файл, который CodeVisionAVR скомпилировала для записи в МК.

Примечание: hex файл находится в папке Папка самой программы/Exe

Файл загрузили. Для записи выберем команду FLASH.Вы увидите окно прогресса, которой будет извещать о проценте записанной программы. После этого произойдет сравнение записанной программы, с программой в буфере обмена. Если все произошло успешно, и вы правильно написали программу, то увидите ее работу на стенде. Если же запись произошла с ошибкой, то повторите все заново, начиная со стирания.

МК рассчитан на 10 000 записей (может быть и меньше)!

Если программатор USB

Если в качестве программатора используется USBAVR910, то после установки его в USB ПК начнет искать драйвер для нового USB устройства. Следует отказаться от автоматической установки и указать путь к папке с драйвером согласно операционной системе (драйвер спросите у преподавателя). Если операционная система не предложила выбрать драйвер, а просто выдала сообщение, что таковой не установлен, то следует перейти в

Диспетчер устройств, где вы увидите неопознанное устройство, которому необходимо обновить драйвер.

После установки у вас появиться новое устройство, которое имеет виртуальный COM-порт (рис. 2.12).

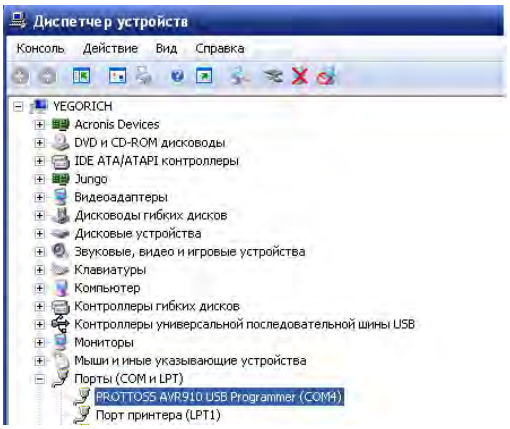



Рис. 2.12. Установка драйвера для программатора

После этого нужно в CodeVisionAvr выбрать именно этот программатор – AtmelAVRProg (AVR910)–и указать номер порта.

CodeVision с USB программатором работает очень медленно (стирание происходит около минуты) из-за того, что он «не умеет» правильно пересылать пакеты данных по протоколу USB. Но существует другая программа, которая предназначена именно для этого программатора – AVRProg, - написанная самим производителем МК (ее тоже можно взять у преподавателя или скачать с сайтов). Это программа входит в пакет AVRStudio. Ее можно просто копировать на другие ПК без пакета AVRStudio.

AVRProg запуститься в случае, если она нашла программатор в списке устройств ПК. Если запуск не произошел, проверьте все контакты. Если она запустилась, то появиться окно, где человек, знающий английский язык, без труда поймет, как с ней обращаться (рис. 2.13).

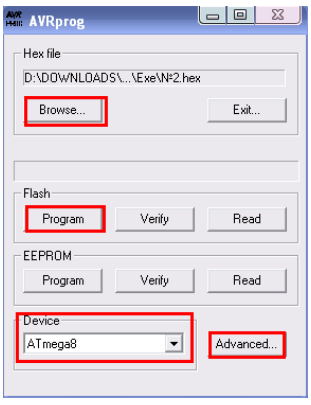



Рис. 2.13. Окно программы AVRprog

## 2.5 Лабораторные работы №1-№4

### Лабораторная работа №1

# ВКЛЮЧЕНИЕ ДВИГАТЕЛЯ И СВЕТОДИОДА (ИНДИКАТОРА) НА ОПРЕДЕЛЕННЫЙ ПРОМЕЖУТОК ВРЕМЕНИ, ИСПОЛЬЗУЯ ПРОГРАММНЫЙ ТАЙМЕР. УСТРАНЕНИЕ ДРЕБЕЗГА КОНТАКТОВ

**Цель работы:**

Научиться использовать порты МК для включения двигателя на определенный промежуток времени, а также для включения светодиодов и реагирования на нажатие кнопки. На основе сведений приведенных ниже устранить дребезг контактов.

**Общие сведения:**

Рассмотрим структуру МК. Порт имеете три части (они же команды в языке С++):

1. DDRx - регистр направления передачи данных - определяет, является тот или иной вывод порта входам или выходом; если некоторый разряд регистра DDRx содержит логический 0, то соответствующий вывод порта сконфигурирован как вход, в противном случае - как выход;
2. PORTx - регистр порта - если вывод выполняет роль выхода, то в соответствующий разряд записывается значение, предназначенное для вывода; если вывод выполняет роль входа, то логический 0 в некотором разряде регистра PORTx соответствует высокоомный вход, а логическая 1 - вход, нагруженный подтягивающим сопротивлением;
3. PINx - регистр выводов порта - в отличие от регистров DDRx и PORTx доступен только для чтения и позволяет считать входные данные порта на внутреннюю шину мк.

В схеме используется кнопка, имеющая одну группу из двух нормально разомкнутых контактов. А если есть контакты, значит, есть и дребезг этих контактов. Теперь рассмотрим способ борьбы с дребезгом контактов программным путем.

Самый простой способ борьбы с дребезгом — введение в программу специальных задержек. Рассмотрим это подробнее. Начнем с исходного состояния, когда контакты кнопки разомкнуты. Программа ожидает их замыкания. В момент замыкания возникает дребезг контактов.

Дребезг приводит к тому, что на соответствующем разряде порта PD вместо простого перехода с единицы в ноль мы получим серию импульсов. Для того, чтобы избавится от их паразитного влияния, программа должна сработать следующим образом. Обнаружив первый же нулевой уровень на входе, программа должна перейти в режим ожидания. В режиме ожидания

программа приостанавливает все свои действия и просто отрабатывает задержку.

Время задержки должно быть выбрано таким образом, чтобы оно превышало время дребезга контактов. Такую же процедуру задержки нужно ввести в том месте программы, где она ожидает отпускания кнопки.

**Ход работы:**

Программирование контроллера AVR в среде CodeVisionAVR. После запуска программы разворачивается окно (рис. 1).

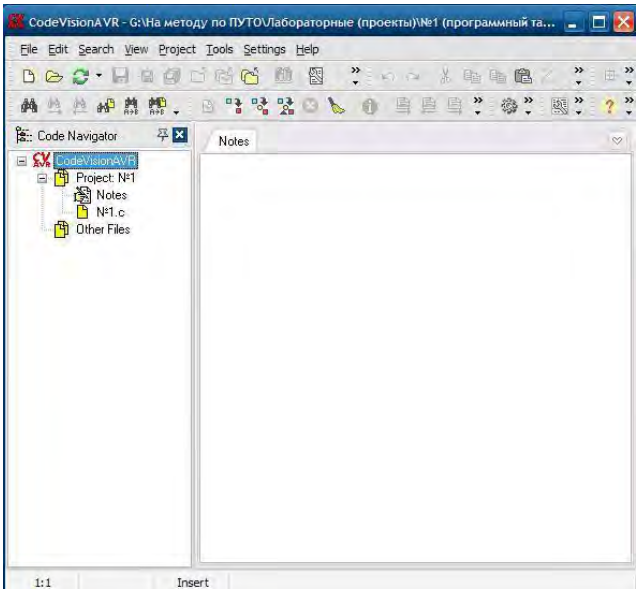



Рис. 1. Окно программы CodeVisionAVR

Далее открываем вкладку FILE, New(рис.2).

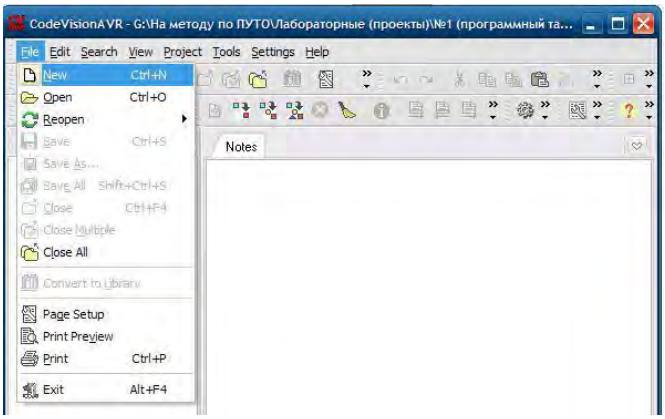



Рис. 2. Создание нового проекта

Выбираем Project (рис.3).

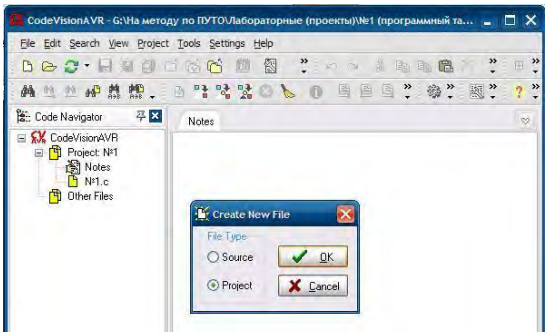



Рис. 3. Выбор проекта в CodeVisionAVR

Дальше спросят, будем ли использовать CodeWizard – соглашаемся (рис. 4).

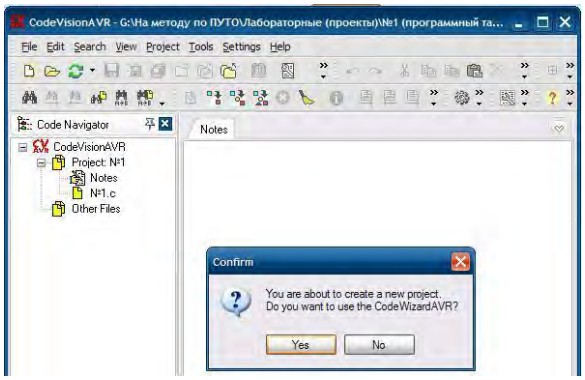



Рис. 4. Выбор мастера CodeWizardAVR

Выбираем в списке МК Chip:ATmega8, и ставим частоту на 8 MHz согласно рис.5.

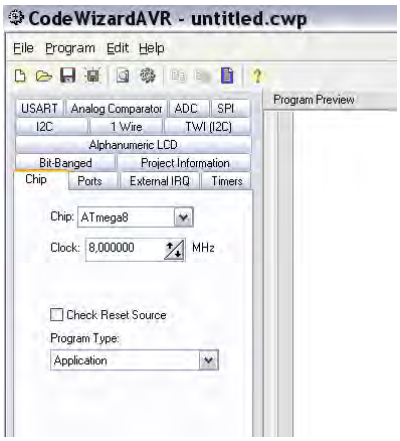



Рис. 5. Окно настройки параметров работы МК Atmega8

Следующим действием выбираем вкладки Ports:PortC, где настраиваем порты С на вход/выход согласно рис.6.

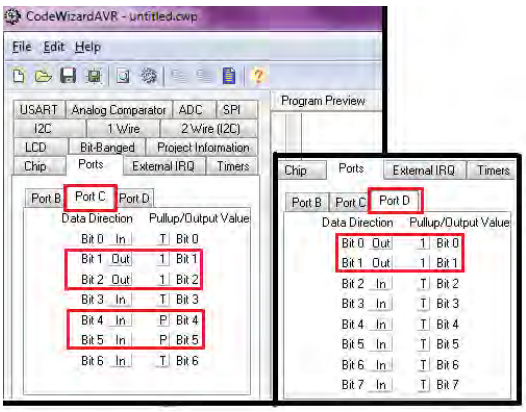



Рис.6. Настройка портов входа/выхода

Это настройка требует пояснений. На выход в данной работе идет 2 порта C.1 и C.2 (соответственно в окне Wizardэто Bit 1 и Bit 2). Чтобы в начале программы выходы были выключены, необходимо выходное состояние поставить в логическую 1 (именно 1, а не 0 – так принято во всех МК в нашем мире). Точно также выставляем порты D.0 и D.1 для включения светодиода.

Порты C.4 и C.5 включаем на вход и включаем у них состояние P - это означает, что к этим портам подключили внутренний подтягивающий резистор для согласования уровней (так выставляется выход на кнопку). Соответственно состояние T – transitозначает отсутствие резистора на выходе.

После того, как все установлено, сохраняем заготовку проекта (рис. 7).

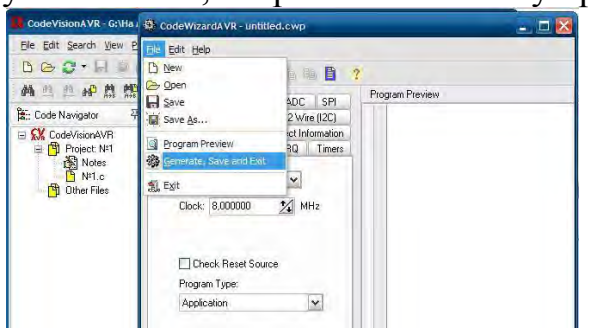



Рис. 7. Компиляция и сохранение проекта

Сохраняем файлы .c .prj .cwp под одним именем и желательно в отдельно созданную папку (рис. 8).

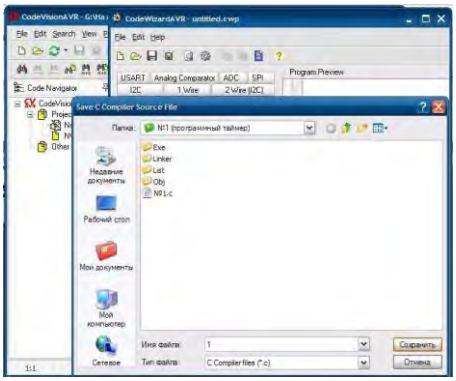


Рис. 8. Сохранение файлов программы

После сохранения открывается окно Си редактора (рис. 9).

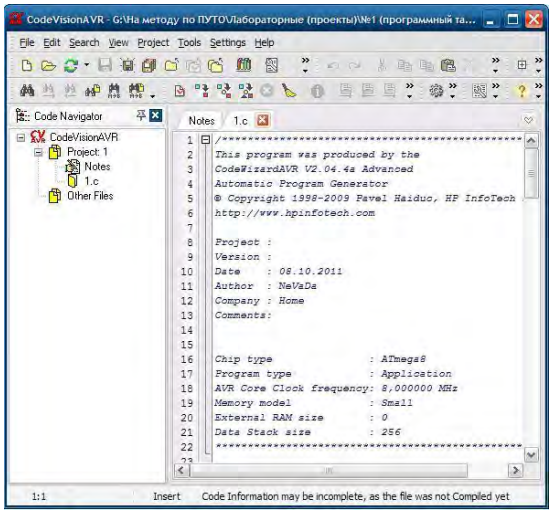



Рис. 9. Окно с листингом программы

Пишем здесь свою программу согласно заданию в цикле while(1). В конце лабораторной работы приведен весь листинг программы с пояснениями.

После того, как программа написана, необходимо ее проверить и сохранить в hex-файл (файл для записи в МК). Для этого компилируем исходный файл (Shift+F9) согласно рис. 10.

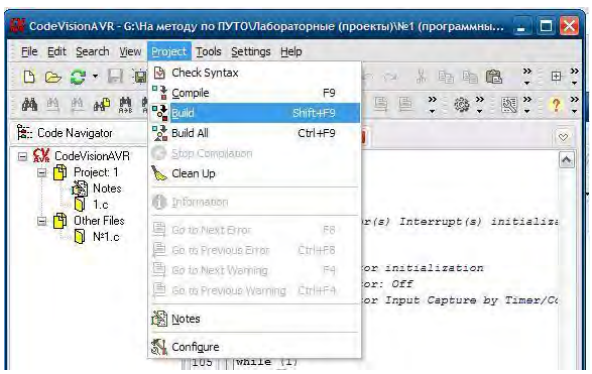



Рис. 10. Компиляция проекта

Нам важно, чтобы после компиляции появилось такое окно с надписями (без ошибок и предупреждений), как на рис.11.

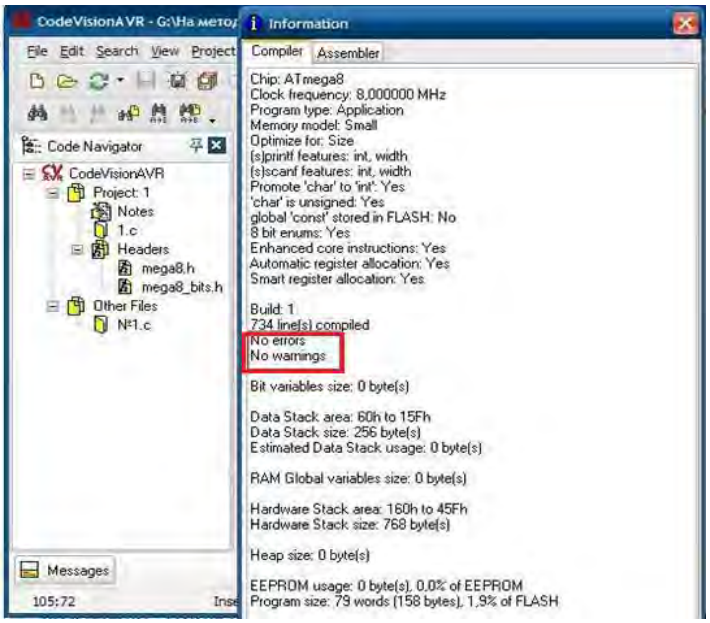



Рис. 11. Окно компиляции в CodeVisionAVR

На рис. 11 в окне компиляции кроме сообщения об ошибках выводится информация о размере сгенерированного кода, проценте занятой памяти при записи на данный МК и прочее.

Примечание: каждый раз, когда вы компилируете набранную программу в CodeVision, автоматически создается hex-файл, который размещен в папке Exe (она размещена в основной папке программы)

**Листинг программы**:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.04.4a Advanced
Automatic Program Generator
© Copyright 1998-2009 PavelHaiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 09.10.2011
Author  :NeVaDa
Company :
Comments:
```

```
Chip type               : ATmega8
Program type            : Application
AVR Core Clock frequency: 8,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

#include<mega8.h>  //библиотека для МКAtmega8 (автоматически)
#include<delay.h>  // подключаем библиотеку задержки

// Declare your global variables here - приглашениеввестикодздесь

voidmain(void)  // начало главной части программы
{
// Declare your local variables here

// Input/Output Ports initialization - настройкапортоввввода/вывода
// Port B initialization   - настройкапортовB
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;  //запись состояний портов типа "вкл/выкл" в 16-м коде
DDRB=0x00;   //запись состояний типа "вход/выход" в 16-м коде

// Port C initialization  -настройкапортовC
// Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out Func0=In
// State6=T State5=P State4=P State3=T State2=1 State1=1 State0=T
PORTC=0x36;
DDRC=0x06;

// Port D initialization  -настройкапортов D
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=0 State1=0 State0=T
PORTD=0x03;  // аналогично для порта D
DDRD=0x03;

// Timer/Counter 0 initialization  -настрокатаймера 0
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;

// Timer/Counter 1 initialization    - настройкатаймера 1
```

```
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization  -настройкатаймера  2
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization  -настройкавнешнихпрерываний
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization   - настройкамаскипрерываний
TIMSK=0x00;

// Analog Comparator initialization   - настройкааналоговогокомпаратора
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
   PORTB.1=1;// при выключенном B.1 включается сигнал для работы двигателя
   while (1)    //основной цикл программы, где и пишем код
        {
   if (PINC.4==0){ //указываем действие при нажатии кнопки 1, подключенной к порту C.4
   delay_ms(200); //задержка для устранения дребезга контактов
         PORTC.1=1;PORTC.2=0; //перключение направления вращения двигателя
         PORTD.0=0; PORTD.1=1; // вкл. 1-й светодиод и выкл. 2-й
   delay_ms(5000);PORTC.2=1; PORTD.0=1;} // выключения светодиода и двигателя по истечении 5 секунд

   if (PINC.5==0) { //аналогично для кнопки 2, подключенной к порту C.5
   delay_ms(200);
         PORTC.2=1;PORTC.1=0;
         PORTD.0=1; PORTD.1=0;
   delay_ms(7000);PORTC.1=1; PORTD.1=1;} // выключения светодиода и двигателя по истечении 7 секунд
        };
   }
```

## Лабораторная работа № 2

# ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ТАЙМЕР ДЛЯ ВКЛЮЧЕНИЯ ДВИГАТЕЛЯ НА ОПРЕДЕЛЕННЫЙ ПРОМЕЖУТОК ВРЕМЕНИ

**Цель работы:**

Изучить принцип работы программного таймера. Написать программу включения двигателя и светодиода, оповещающего о направлении вращения вала двигателя, на определенный промежуток времени.Выяснить преимущества программного таймера.

**Общие сведения:**

Таймер/Счетчик0 – это самый простой таймер или счетчик в семействе микроконтроллеров AT. У него нет дополнительных функций, он просто считает.

Почему называется «таймер/счетчик». На самом деле Таймер/счетчик – аппаратная часть МК, которая подсчитывает количество каких-либо импульсов. Таких импульсов может быть два вида:

1. Генератор МК;
2. Импульсы на ножке МК.

1. Генератор МК. МК работает с определенной частотой, которая определяется выбранным генератором (внутренним или внешним). Так вот, в этом режиме импульс для таймера подается от этого генератора с той же частотой. Именно этот режим работы таймера/счетчика и называется Таймером.

2. Импульсы на ножке МК. Значение таймера/счетчика изменяется в зависимости от состояния сигнала на определенной ножке. Т.е. считает количество импульсов на ножке. Этот режим называют Счетчиком.

Как же работает Таймер/Счетчик? В памяти МК есть часть отведенная для таймера, называется регистр TCNT0. Этот регистр, при запуске равен 0, при появлении импульса от генератора или ножки МК, значение регистра увеличивается на 1. Так как регистр TCNT0 восьмиразрядный, то его максимальное значение равно 0xFF. После того как Таймер/Счетчик насчитал 256 импульсов (0xFF) он сбрасывается на 0 и начинает все заново.

Таймер/Счетчик0 при переполнении и сбросе регистра вызывает TCNT0 прерывание. С какой частотой будет выполняться прерывание не сложно посчитать. Для этого частоту работы МК надо разделить на 256 (максимальное значение таймера). К примеру, частота ATmega8 по умолчанию (заводские настройки) составляет 1МГц, отсюда 1000000/256= 3906,25 раз в секунду будет вызываться прерывание. Это много! Для этого

существуют предделители. Предделитель заставляет таймер реагировать не на каждый импульс, а через n раз.

В ATmega8 для Таймера/Счетчика0 можно устанавливать предделители 8, 64, 256, 1024. Таким образом таймер0 будет считать не все импульсы подряд, а только 8-ой, 64-ый, 256-ой и 1024-ый соответственно. Посчитаем на примере, 1000000/8/256= 488,28125 Герц. – это с предделителем 8. и 1000000/1024/256= 3,814697265625 в секунду с предделителем 1024.

Таким образом, зная частоту работы МК можно высчитывать частоту срабатывания прерываний. Такое прерывание называется «прерывание по переполнению таймера0».

Но что делать, если надо выставить конкретную частоту, которую нам не могут обеспечить предделители. Для этого существует «прерывание при совпадении с регистром (обычно A или B). В соответствующий регистр записываем любое число в зависимости от разрядности в 16-м коде, по достижении которого таймером, происходит выполнение какой-либо операции.

Одна трудность: предделитель нельзя использовать в режиме счетчика, только если таймер работает от генератора МК. Это плохо, но для подобных целей используются другие таймеры/счетчики.

У каждого МК есть определенное количество таймеров, например у Atmega8 существует только 3:

1. Timer0 – регистров сравнения нету;
2. Timer1 - два регистра сравнения (16-разрядных);
3. Timer2 - один регистр сравнения (8-разрядный).

**Ход работы:**

Выберем МК и частоту его работы, как и прошлой работе. PORT C.1 и C.2ставим на выход (рис. 1). Кнопок в этой работе использовать не будем, ввиду некоторой сложности работы.

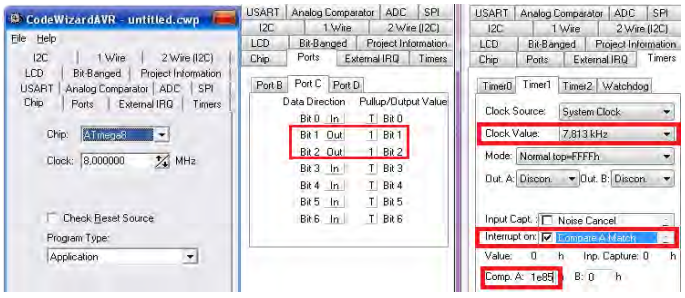



Рис.1. Настройка портов и таймера МК при помощи Wizard

Кроме включения двигателя будем включать светодиоды, которые будут извещать о направлении вращения вала двигателя. Для этого порт D сконфигурируем так: BitD1- Out – 1; BitD3 - Out – 1;

В параметрах таймера выберем

Timer 1,

Clock Value 7,813 kHz, - частота счета
Interrupton: Compare A Match, - прерывание по совпадению с регистром А.
Comp. A = 1e85 (это число означает, что он будет тактироваться с частотой в 1 секунду, т.к мы выставили прерывание, когда таймер насчитает7813).

Генерируем код и сохраняем.
В листинге, кроме известных, появилась еще новая строка:
interrupt [TIM1_COMPA] void timer1_compa_isr(void) . Здесь надо вписать действие при прерывании таймера:

```
TCNT1H=0;
TCNT1L=0;
sec++;
```

Для корректной роботы таймера в эти регистры обязательно записать 0 (так таймер будет сбрасываться при достижении 7813 и считать заново). И еще в «тело» таймера добавили переменную, которую будем инкрементировать:sec++ (она и будет для нас задержкой в секундах).

**Листинг программы (с пояснениями)**:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.04.4a Advanced
Automatic Program Generator
© Copyright 1998-2009 PavelHaiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 09.10.2011
Author  :NeVaDa
Company :
Comments:

Chip type               : ATmega8
Program type            : Application
AVR Core Clock frequency: 8,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

#include <mega8.h>

unsigned int sec; // Объявляем переменную sec типом - беззнаковый целый
```

```
// Здесь указываем действие выполняемое при прерывания таймера
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
TCNT1H=0; // обязательно нужно выставить регистры в 0
TCNT1L=0; // их два, т.к. таймер 16-битный
sec++; // прибавляем к переменной sec 1.
}

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out Func0=In
// State6=T State5=P State4=P State3=T State2=1 State1=1 State0=T
PORTC=0x36;
DDRC=0x06;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=1 State1=1 State0=T
PORTD=0x06;
DDRD=0x06;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 7,813 kHz
// Mode: Normal top=FFFFh
```

```
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x05;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x1E; // это регистр сравнения A, при совпадении с которым таймер
OCR1AL=0x85; // выполняет описанное выше действие. В итоге получаем частоту 1 Гц.
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
// Global enable interrupts
#asm("sei")

PORTB.1=1;// при выключенном B.1 включается сигнал для работы двигателя
while (1)
    {
if (sec==0){PORTC.1=0;PORTD.1=0;} // в начале времени вкл. порт C.1 - вращение вала двигателя + вкл. 1-го диода на 5 сек
if (sec==5){PORTC.1=1;PORTD.1=1;} // выключаем порт C.1 - остановка двигателя и выкл.1-го диода на 2 секунды
if (sec==7){PORTC.2=0;PORTD.3=0;} // вкл. порт C.2 - вращение вала двигателя в другую сторону + вкл.3-го диода на 7 сек
if (sec==14) {PORTC.2=1;PORTD.3=1; sec=0;} // выкл. порт C.2 - ост.двигателя + выкл.3-го диода, возврат в начало цикла
      };
}
```

Как видно из проделанной работы, для наших целей не очень удобно было использовать аппаратный таймер. Но именно аппаратный таймер позволяет разгрузить процессор, саму программу и позволяет более точно делать временные задержки. С использованием аппаратного таймера можно делать задержки на часы и даже на сутки и при этом получить погрешность от 1 сек (за час) до 18 секунд (за сутки). Эти погрешности связанны с тем, что в основе генератора МК положена работа RC-генератора, погрешность у которого ±3%. Чтобы сделать таймер/часы точнее нужно подключать к МК внешний кварцевый генератор.

Что же касается программного таймера, то точность у него очень мала (±20%). Поэтому его целесообразно использовать для кратковременного включения чего-либо.

## Лабораторная работа №3

# ПРОГРАММНАЯ ШИРОТНО-ИМПУЛЬСНАЯ МОДУЛЯЦИЯ (ШИМ) ДЛЯ УПРАВЛЕНИЯ МОЩНОСТЬЮ ДВИГАТЕЛЯ

**Цель работы:**

Ознакомиться с принципом ШИМ. На основе полученных знаний написать программу, в которой при нажатии первой кнопки скорость вращения вала двигателя будет возрастать от min к max, а при нажатии второй кнопки скорость вращения вала двигателя будет снижаться. При этом диоды должны извещать о скорости вращения вала двигателя по закону:

- включен 1 диод - скорость 25% от max скорости вращения вала;
- включено 2 диода – скорость 50 %;
- включено 3 диода – скорость 75%;
- включено 4 диоды – скорость 100%.

**Общие сведения:**

Широтно-импульсная модуляция (в зарубежных источниках этот режим называется PWM - PulseWidthModulation) - способ задания аналогового сигнала цифровым методом, то есть из цифрового выхода, дающего только нули и единицы, позволяет получить плавно меняющиеся величины.

Представим себе мощный двигатель. Если включить его постоянно, то он раскрутится до максимального значения и так будет работать. Если выключить, то остановится через некоторое время, вследствие своей инерционности. А вот если двигатель включать на несколько секунд каждую минуту, то он раскрутится, но далеко не на полную скорость - большая инерция сгладит рывки от включающегося двигателя, а сопротивление от трения не даст ему крутиться бесконечно долго. Чем больше продолжительность включения двигателя в минуту, тем быстрей он будет крутиться.

При ШИМ подается на выход сигнал, состоящий из высоких и низких уровней (применимо к нашей аналогии — включаем и выключаем двигатель), то есть нулей и единицы. А затем это все пропускается через интегрирующую цепочку (в нашем случае драйвер L293D и масса якоря двигателя). В результате интегрирования на выходе будет величина напряжения, равная площади под импульсами.

Меняя скважность (отношение длительности периода к длительности импульса) можно плавно менять эту площадь, а значит и напряжение на выходе (рис. 1). Таким образом, если на выходе сплошные 1, то на выходе будет напряжение высокого уровня, в случае нашего двигателя, на выходе из моста L293 это 5 вольт, если нули, то ноль. А если 50% времени будет высокий уровень, а 50% низкий то 2,5 вольт. Интегрирующей цепочкой тут

будет служить масса якоря двигателя, обладающего малой, но достаточной инерцией.


Рис. 1. Схема ШИМ сигнала

А если взять и подавать ШИМ сигнал не от нуля до максимума, а от минуса до плюса. Скажем от +12 до -12. А можно задавать переменный сигнал! Когда на входе ноль, то на выходе -12В, когда один, то +12В. Если скважность 50% то на выходе 0В. Если скважность менять по синусоидальному закону от максимума к минимуму, то получим переменное напряжение. А если взять три таких ШИМ генератора и подавать через них синусоиды сдвинутые на 120 градусов между собой, то получим самое обычное трехфазное напряжение, а значит сможем управлять бесколлекторными асинхронными и синхронными двигателями. На этом принципе построены все современные промышленные привода переменного тока типа Unidrive и OmronJxx.

В качестве сглаживающей интегрирующей цепи в ШИМ применена обычная RC цепочка (в нашем случае она включена в микросхему L293D) и масса якоря двигателя.

Существует аппаратная и программная ШИМ. Разница заключается в точности, сложности настройки, а также в том, что аппаратной ШИМ у МК ограниченное количество (обычно 1-4 канала), а программной ШИМ можно реализовывать столько, сколько ножек у МК.

В этой лабораторной работе научимся использовать программную ШИМ.

Для задания мощности двигателя выведем формулу:

$$P_{дв} = \frac{t_{импульса}}{t_{скважности}} \cdot 100\%,$$

где$P_{дв}$ - мощность двигателя от номинальной в процентах,

$t_{импульса}$ – длительность 1 импульса,

$t_{скважности}$ – длительность скважности.

Для того, чтобы задать программную ШИМ, достаточно включить нужный порт на время $= t_{импульса}$ и выключить на время $= t_{скважности}$, в зависимости от их соотношения и получим необходимую мощность двигателя.

**Ход работы:**

При помощи мастера CodeWizard (Shift+F2) выставить порты согласно таблице 1 или рис. 2.

Таблица 1.

Настройка портов МК

| Имя и номер порта | Вход/выход | включен/выключен ( в начале программы) |
|---|---|---|
| B.1 | Выход | вкл. |
| C.1 | Выход | выкл. |
| C.2 | Вход | выкл. |
| C.4 | Вход | вкл. |
| C.5 | Выход | вкл. |
| D.0 | Выход | выкл. |
| D.1 | Выход | выкл. |
| D.2 | Выход | выкл. |
| D.3 | Выход | выкл. |

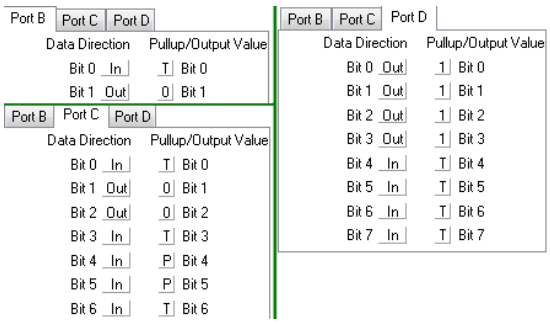



Рис. 2. Настройка портов МК

Получить программу исходя из алгоритма: есть переменная, которая в зависимости от нажатий кнопок меняет свои значения, эти значения и будут определять мощность и включение светодиодов. В качестве таймера использовать команду delay, т.е. включать порт B.1, к примеру, на 1 мкс и выключать на 10 мкс для задания 10% от номинальной мощности. Написать программу согласно заданию!

**Листинг основной части программы:**

```
#include<mega8.h>
#include<delay.h>
unsignedintvar; // описываем переменную типа безнаковый целый
……
```

```
……
PORTC.1=0; // прописываем направление вращения двигателя
var=1; // присваиваем начальное значение
while (1)
{
if (PINC.4==0 &var!=4) {delay_ms(200); var++;} //
действиепринажатиикнопкиC.4
if (PINC.5==0 &var!=1) {delay_ms(200); var--;} //
действиепринажатиикнопкиC.4
if (var==1) {PORTD=0x0E; PORTB.1=1; delay_ms(6);
            PORTB.1=0; delay_ms(24); }        // 25% отmaxскорости
if (var==2) {PORTD=0x0C; PORTB.1=1; delay_ms(12);
            PORTB.1=0; delay_ms(24); }        // 50% отmaxскорости
if (var==3) {PORTD=0x08; PORTB.1=1; delay_ms(18);
            PORTB.1=0; delay_ms(24); }        // 75% отmaxскорости
if (var==4) {PORTB.1=1; PORTD=0x00;}       // max скорость
};}
```

При нажатии кнопки button1 увеличивается значение переменной var на 1, до значения 4 с задержкой в 200 мс (это ещё и для устранения дребезга контактов). При нажатии кнопки button2 уменьшается значение переменной var на 1, до значения 1 с задержкой в 200 мс. В итоге в зависимости от значения var порт B.1 включается и выключается на промежутки времени, соотношение которых и дает нам нужную мощность. Также значение var влияет на диоды включение, которых извещает нам о мощности.

Появилась одна тонкость: вместо того, чтобы при значении var=1 на включение светодиодов вместо записи

PORTD.0=0; PORTD.1=1; PORTD.2=1; PORTD.3=1; // включить 1 светодиод, подключенный к порту D.1

можно записать это в 16-м коде типа

PORTD=0x0E // в 2-м коде это 11110, где предпоследняя единица нужна из-за особенностей структуры МК, а остальные 1110 указывают как раз вкл/выкл портов по убыванию (в данном случае от D.3 до D.0соответственно).

Такой вариант записи сокращает программу, что может пригодится при написании программ большего размера, чем флэш-память МК. Для перевода величин использовался инженерный калькулятор в Windows, но есть ещё способ: выбрать вкл./выкл. портов в CodeWizard и затем скопировать нужные строки (рис.3).

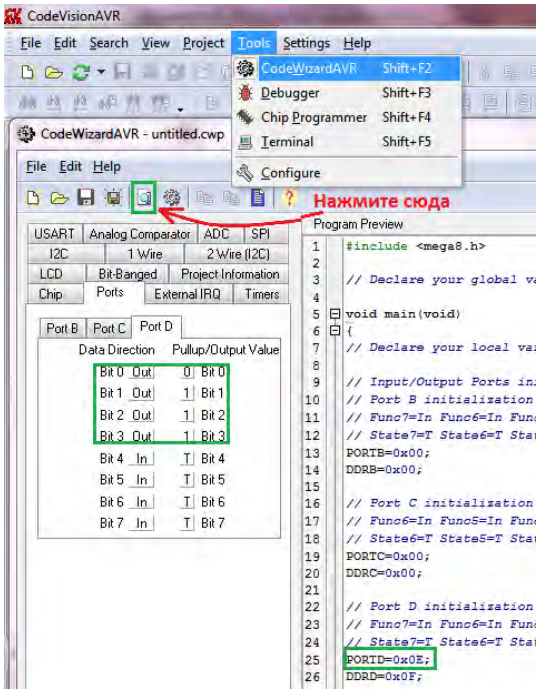



Рис.3. Просмотр созданной программы в CodeWizard

Как видно из программы, скорость вращения вала двигателя будет переключаться мгновенно, а не плавно, как это можно сделать при помощи аппаратной ШИМ. Это является основным недостатком программной ШИМ.

**Примечания к** Proteus

Для наблюдения изменения сигнала воспользуйтесь осцилографом.

## Лабораторная работа №4

# ИСПОЛЬЗОВАНИЕ АППАРАТНОЙ ШИМ ДЛЯ УПРАВЛЕНИЯ ДВИГАТЕЛЕМ

**Цель работы:**

Изучить принцип работы внутренних регистров МК и написать программу изменения мощности двигателя при помощи аппаратной ШИМ.

**Общие сведения:**

Аппаратную ШИМ проще всего сделать на ШИМ генераторе, который встроен в таймеры МК. В Atmega8 3 ШИМ канала (рисунок 1):

- OC1A (16-битный);
- OC1B (16-битный);
- OC2 (8-битный).

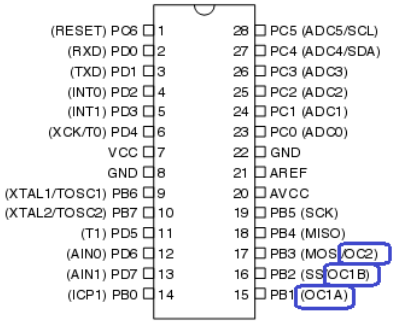



Рис.1. ШИМ каналы МК Atmega8

У таймера есть особый регистр сравнения OCR**. Когда значение в счётном регистре таймера достигает значения находящегося в регистре сравнения, то могут возникнуть следующие аппаратные события:

1. Прерывание по совпадению (уже использовали в аппаратном таймере)
2. Изменение состояния внешнего выхода сравнения OC**.

Предположим, что мы настроили наш ШИМ генератор так, что, когда значение в счетном регистре больше, чем в регистре сравнения, то на выходе у нас 1, а когда меньше, то 0. Таймер будет считать как ему и положено, от нуля до 256, с частотой, которую мы настроим битами предделителя таймера. После переполнения сбрасывается в 0 и продолжает заново. В итоге на выходе появляются импульсы. А если увеличить значение в регистре сравнения, то ширина импульсов станет уже.

У таймера может быть определенное количество регистров сравнения. Зависит от модели МК и типа таймера. Например, у Atmega8:

4. Timer1 - два регистра сравнения (16-разрядных)

5. Timer2 - один регистр сравнения (8-разрядный)

Самих режимов ШИМ существует несколько (рис. 2, 3):

a. Fast PWM (быстрый ШИМ)

В этом режиме счетчик считает от нуля до 255, после достижения переполнения сбрасывается в нуль и счет начинается снова. Когда значение в счетчике достигает значения регистра сравнения, то соответствующий ему вывод OCxxсбрасыватся в ноль. При обнулении счетчика этот вывод устанавливается в 1.

Частота получившегося ШИМ сигнала определяется просто: частота процесора 8МГц, таймер считает до 256 с тактовой частотой. Значит один период ШИМ будет равен $8\,000\,000/256 = 31250$Гц. Это максимальная скорость на внутреннем 8Мгц тактовом генераторе. Еще есть возможность повысить разрешение, сделав счет 8, 9, 10 разрядным (если разрядность таймера позволяет), но надо учитывать, что повышение разрядности, вместе с повышением дискретности выходного аналогового сигнала, резко снижает частоту ШИМ.

b. PhaseCorrect PWM (ШИМ с точной фазой)

Работает также, но счетчик считает уже по-другому. Сначала от 0 до 255, потом от 255 до 0. Вывод OCxx при первом совпадении сбрасывается, при втором устанавливается.

Но частота ШИМ при этом падает вдвое, из-за большего периода. Основное его предназначение, делать многофазные ШИМ сигналы, например, трехфазную синусоиду. Чтобы при изменении скважности не сбивался угол фазового сдвига между двумя ШИМ сигналами. Т.е. центры импульсов в разных каналах и на разной скважности будут совпадать. Чтобы не было кривых импульсов, то в регистр сравнения любое значение попадает через буферный регистр и заносится только тогда, когда значение в счетчике достигнет максимума. Т.е. к началу нового периода ШИМ импульса.

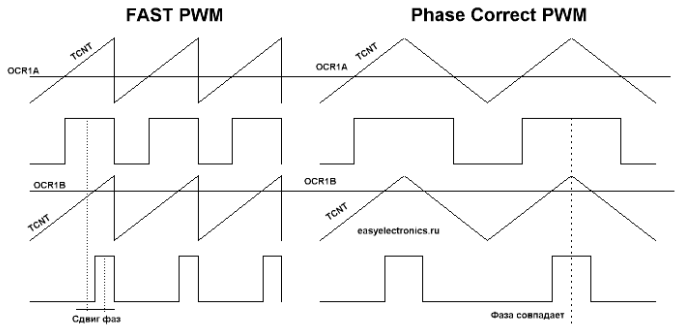



Рис.2 Сравнение режимов Fast PWM и Phase Correct PWM аппаратной ШИМ

c. Clear Timer On Compare (Сброс при сравнении)

Это уже скорей ЧИМ - частотно-импульсномоделированный сигнал. Тут работает несколько иначе, чем при других режимах. Тут счетный таймер считает не от 0 до предела, а от 0 до регистра сравнения! А после чего сбрасывается.

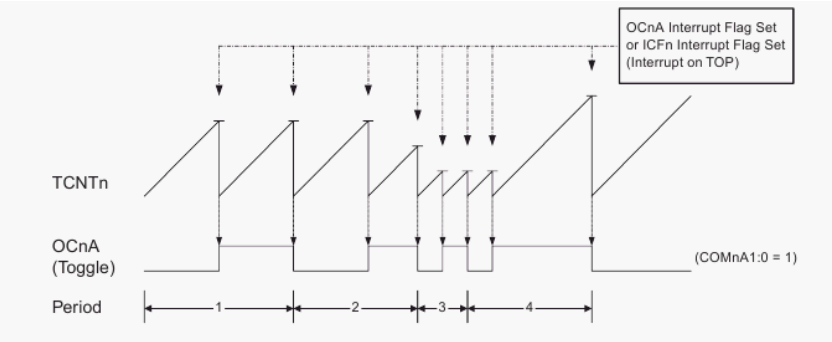



Рис. 3. Режим Clear timer on compare (TCNT) ШИМ МК Atmega8

В лабораторной работе вы научитесь использовать «быстрый ШИМ». Он относительно прост в настройке и удобен для того, чтобы снизить мощность двигателя.

**Ход работы:**

- Создать проект при помощи CodeWizard. Выставить порты согласно таблице 3.1 или рисунку 3.2 предыдущей лабораторной работы.
- Во вкладке Timer1 настроить таймер на работу ШИМ. Для этого необходимо выставить режимы согласно рис.4.

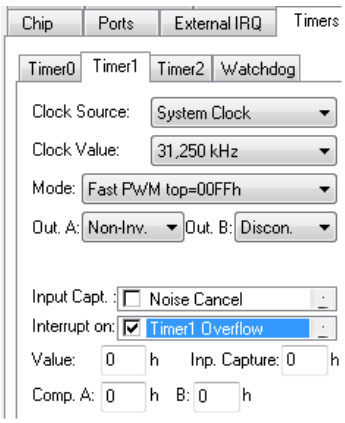



Рис.4. Настройка таймера на режим Fast PWM аппаратной ШИМ

В итоге получаем таймер, который будет считать с частотой 31,25 кГц (как показала практика, чем меньше частота, тем лучше работает МК, также частота влияет на период импульса) и выдавать на выход А (т.е. порт B.1) не инвертирующий сигнал ШИМ (значение Non-inv). После того, как таймер

досчитает до 255, он сброситься в ноль и будет считать заново (значение Interrupton:Timer1 Overflow).

- Получить программу аналогичную программе лабораторной работы №3, но уже с использованием аппаратного таймера.

**Листинг основной части программы:**

```
PORTC.1=0; // задаем направление вращения
while (1)
      {
  if((PINC.4==0)&(OCR1A!=255)) // увеличивать пока не достигло максиума
{  delay_ms(7); // задержка 7 мс (скорость наростания уровня сигнала)
     OCR1A++; //увеличиваем заполнение
  }
  if((PINC.5==0)&(OCR1A!=0)) // уменьшать пока не достигло миниума
  {
  delay_ms(7); // задержка 7 мс (скорость снижения уровня сигнала).
  OCR1A--; //уменьшаем заполнение
        };
if (OCR1A>=64) PORTD.0=0; else PORTD.0=1;  // вкл/выкл 1-й светодиод
if (OCR1A>=85) PORTD.1=0; else PORTD.1=1;  // вкл/выкл 2-йсветодиод
if (OCR1A>=128) PORTD.2=0; else PORTD.2=1; // вкл/выкл 3-йсветодиод
if (OCR1A>=254) PORTD.3=0; else PORTD.3=1; // вкл/выкл 4-йсветодиод
} }
```

При нажатии кнопки button1 ширина импульса ШИМ сигнала (OCR1A, поступающего на порт B.1) увеличивается со скоростью 1мкс за 7 мс до тех пор, пока не достигнет максимума (значения 255). При нажатии кнопки button2 возникает обратное действие до тех пор, пока ширина ШИМ сигнала не станет минимальной (значение 0), т.е. 7 мкс (это было установлено опытным путем).

Включение светодиодов делаем исходя из сравнения величины сигнала в регистре OCR1A с величинами, характеризиющими скорость вращения вала. Например, для 25% это будет величина равная

$$25\% \cdot 255 = 63{,}75 \approx 64$$

Исходя из проделанных работ, можно выявить ещё одно преимущество аппаратной ШИМ – это ее плавность изменения, чего нельзя добиться программным путем.

**Примечания к** Proteus

Для наблюдения изменения сигнала воспользуйтесь осцилографом.

# 3. ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРОГРАММИРОВАНИЮ МИКРОКОНТРОЛЛЕРОВ PIC

## Лабораторная работа 1

## Изучение работы учебного стенда НТЦ-02.31.2

**Цель работы:**

Изучить состав, функциональную схему стенда, ознакомиться со структурой и принципом работы микроконтроллера, изучить порядок работы стенда.

**Порядок выполнения работы:**

3 Изучить краткие теоретические сведения о структуре и принципе работы микроконтроллера.
4 Изучить функциональную схему стенда.
5 Изучить порядок работы со стендом.
6 Изучить расположение и назначение рабочих органов стенда и разъёмов для подключения внешних устройств.
7 Оформить отчёт по лабораторной работе.
8 Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Микроконтроллеры семейства dsPIC33fj32

Микроконтроллер dsPIC33fj32mc204 относится к семейству 16-ти разрядных Flash микроконтроллеров с поддержкой команд цифровой обработки сигналов. Высокое быстродействие (40 MIPS для dsPIC33FJ) и эффективная система команд позволяет использовать контроллеры в сложных системах реального времени. Ключевые особенности:

- расширенная система команд, включающая специфические команды поддержки цифровой обработки сигналов (DSP).
- 24-разрядные инструкции выполняются за 2 периода тактовой частоты, за исключением команд деления, переходов, команд пересылки данных из регистра в регистр и табличных команд.
- разрядность программного счетчика позволяет адресовать до 4M слов программной памяти (4M * 24бит).
- аппаратная поддержка циклов типа DO и REPEAT, выполнение которых не требует дополнительных издержек программной памяти и времени на анализ условий окончания, в то же время эти циклы могут быть прерваны событиями прерывания в любой момент;
- 16 рабочих регистров, каждый регистр массива может выступать как данные, адрес или смещение адреса

- два класса команд: микроконтроллерные инструкции (MCU) и команды цифровой обработки сигналов (DSP). Оба этих класса равноправно интегрированы в архитектуру контроллера и обрабатываются одним ядром.
- различные типы адресации;
- система команд оптимизирована для получения максимальной эффективности при программировании на языке высокого уровня Си.

Микроконтроллер обладает обширным набором периферийных модулей:

- порты ввода-вывода общего назначения;
- таймеры;
- модули захвата и сравнения;
- модули генерации ШИМ сигналов;
- модуль интерфейса квадратурного энкодера;
- 10- и 12- битный аналого-цифровой преобразователь;
- коммуникационные интерфейсы (UART, SPI, I2C, DCI, CAN).

Структурная схема микроконтроллера dsPIC33fj32mc204 представлена на рисунке 1.1. Основные параметры приведены в таблице 1.1. Цоколёвка микроконтроллера приведена в приложении А.

Таблица 1.1

Основные параметры микроконтроллера

| | |
|---|---|
| ЦПУ: Ядро | dsPIC33 |
| ЦПУ: F, МГц | от 0.5 до 7.37 |
| Память: Flash, Кбайт | 32 |
| Память: RAM, Кбайт | 2 |
| I/O (макс.), шт. | 35 |
| Таймеры: 16-бит, шт | 3 |
| Таймеры: Каналов ШИМ, шт | 2 |
| Таймеры: RTC | Нет |
| Интерфейсы: UART, шт | 1 |
| Интерфейсы: SPI, шт | 1 |
| Интерфейсы: I2C, шт | 1 |
| Аналоговые входы: Разрядов АЦП, бит | 10 |
| Аналоговые входы: Каналов АЦП, шт | 9 |
| Аналоговые входы: Быстродействие АЦП, kSPS | 1100 |
| $V_{CC}$, В | от 3 до 3.6 |
| $I_{CC}$, мА | 250 |
| $T_A$, °C | от -40 до 125 |
| Корпус | TQFP-44 |

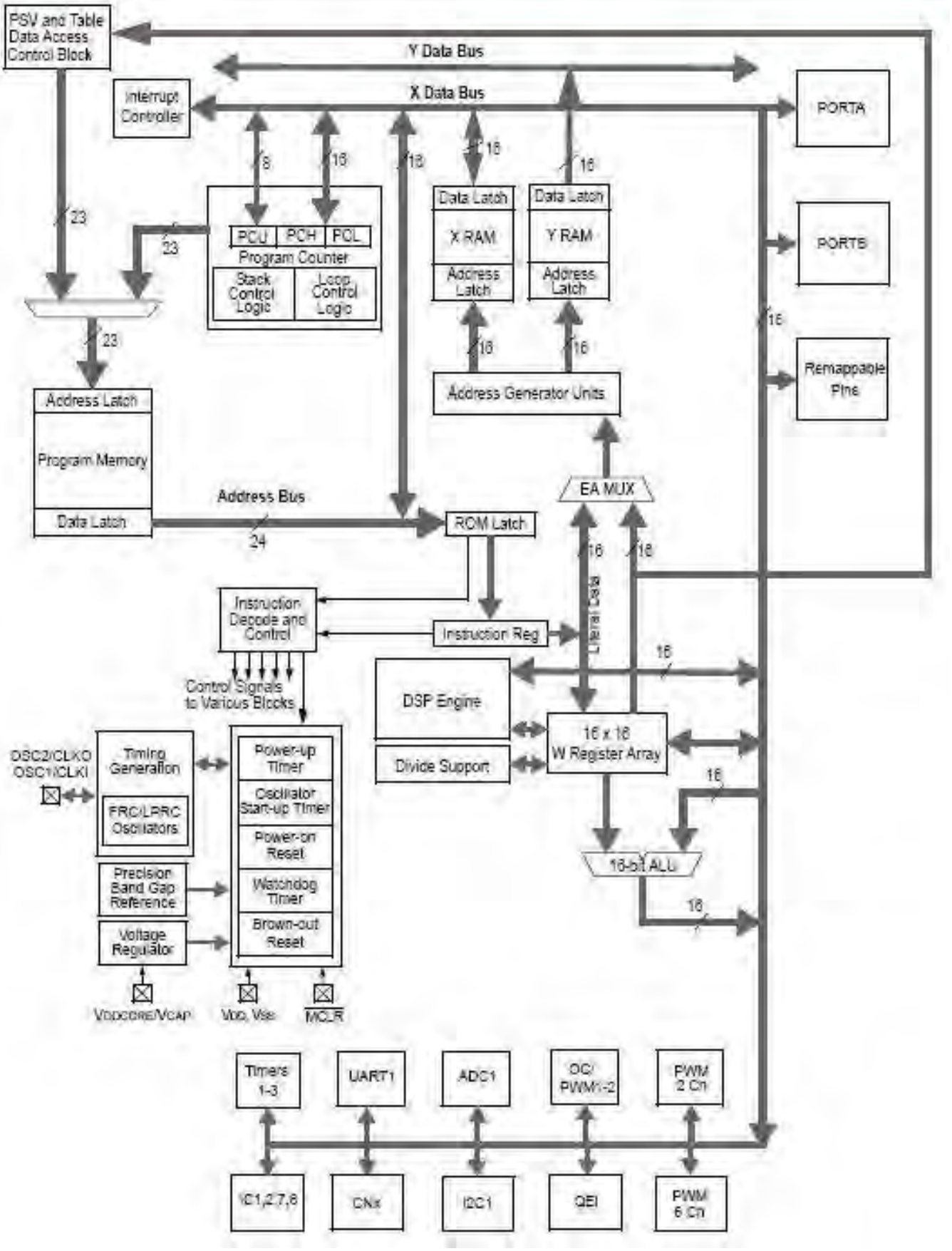



Рис. 1.1. Структурная схема микроконтроллера

## 1.2 Среда разработки MPLAB IDE

Основной средой разработки для всех семейств (и 8- и 16-битных) является MPLAB IDE, которую предоставляет компания Microchip. Среда имеет следующие встроенные средства и возможности:

- менеджер проекта и рабочей области.
- текстовый редактор с подсветкой кода для ассемблеров всех семейств и языка Си. Возможен парсинг кода и вывод списка элементов программы (функций, переменных, констант, структур, и т. п.). Текстовый редактор поддерживает запись и выполнение макросов.
- ассемблер для всех семейств Microchip. Для 16-битных контроллеров - ассемблер ASM30. В состав среды входят заголовочные файлы и скрипты линкера для всех контроллеров.
- программный симулятор для всех семейств Microchip, который обеспечивает симуляцию большинства периферийных модулей, трассировку выполняемого кода, произвольное число точек останова, пошаговое выполнение. Симулятор имеет счетчик времени выполнения, логический анализатор (графическое представление изменений логических уровней на выводах контроллера).

Симулятор позволяет имитировать работу внешних устройств: аналоговый сигнал на входе АЦП, RS-232 терминал, и т. п. Возможно изменение любого служебного регистра или ячейки ОЗУ в соответствии с заданной в текстовом файле последовательностью.

Для 16-битных семейств компания Microchip предлагает компилятор MPLAB C30, в состав которого входят: собственно компилятор исходных файлов в объектные; линкер; генератор библиотечных файлов.

Основные особенности компилятора C30:

1) ANSI C совместимый, в комплекте поставляются стандартные библиотеки.
2) поддержка типов 32-bit double и 64-bit double, 64-битного целого (long long).
3) поддержка всех 16-битных семейств Microchip.
4) поддержка смешанного кода Си + asm.
5) поддержка расширенных настроек оптимизации кода.
6) наличие встроенных (intrinsincs) функций для работы с DSP-ядром.

### **Учебный стенд НТЦ**-02.31.2

1.3.1 Структура стенда

Структурная схема стенда представлена на рисунке 1.2. Электрическая схема стенда представлена в приложении Б.

Стенд построен на базе микроконтроллера dsPIC33fj32mc204. Микроконтроллер имеет встроенное ОЗУ для хранения пользовательских программ. Для изучения простейших операций ввода-вывода дискретных данных и ввода аналоговых данных непосредственно к контроллеру подключены светодиодные индикаторы, тумблеры и аналоговые задатчики. Для изучения принципа обработки квадратурного сигнала используется

энкодер. Для подключения большого числа периферийных модулей в стенде организована последовательная шина данных, по которой происходит управление портами расширений дискретных входов-выходов, а так же светодиодной семисегментной индикацией. Так же для заимодействия с пользователем посредством знако-символьной информации к контроллеру подключён жидко-кристаллический индикатор (ЖКИ). С помощью дешифратора и мультиплексора реализована возможность обработки клавиатуры, построенной по матричной схеме. Для подключения иных периферийных устройств (ЦАП, внешняя память) используется шина $I^2C$. Для организации связи с внешними устройствами (в частности с ПК) микроконтроллер имеет порт последовательного асинхронного приёмо-передатчика, позволяющий организовывать обмен данными по интерфейсу RS232.

Все устройства, входящие в состав стенда и отображённые на структурной схеме стенда, являются программно-доступными.

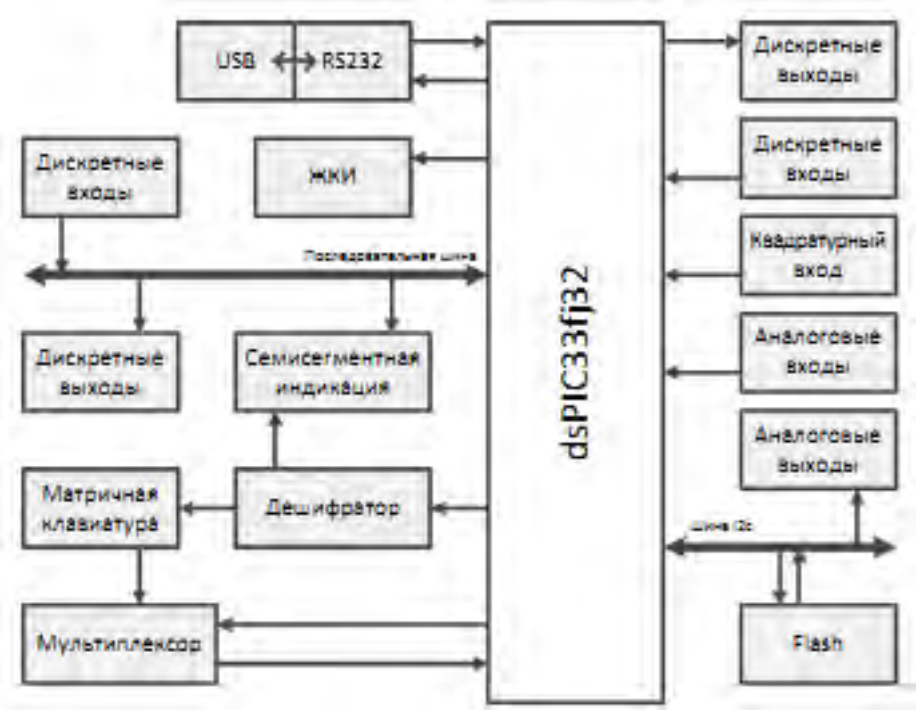



Рис. 1.2. Структурная схема учебного стенда

### 1.3.2 Органы управления

Лицевая панель стенда представлена на рисунке 1.3. На передней панели стенда расположены:

1 – датчики дискретных сигналов (10 переключателей SA1..SA10);

2 – дискретный светодиодный индикатор (10 светодиодов VD1..VD10); 3 – светодиодный семисегментный индикатор (LED);

4 – матричный жидкокристаллический индикатор (LCD);

5 – индикатор выходного аналогового сигнала (LLI);
6 – датчики аналоговых сигналов (2 задатчика RP1, RP2);
7 – источник квадратурного сигнала (энкодер EP1);
8 – матричная двенадцатикнопочная клавиатура;
9 – кнопка сброса контроллера RESET;

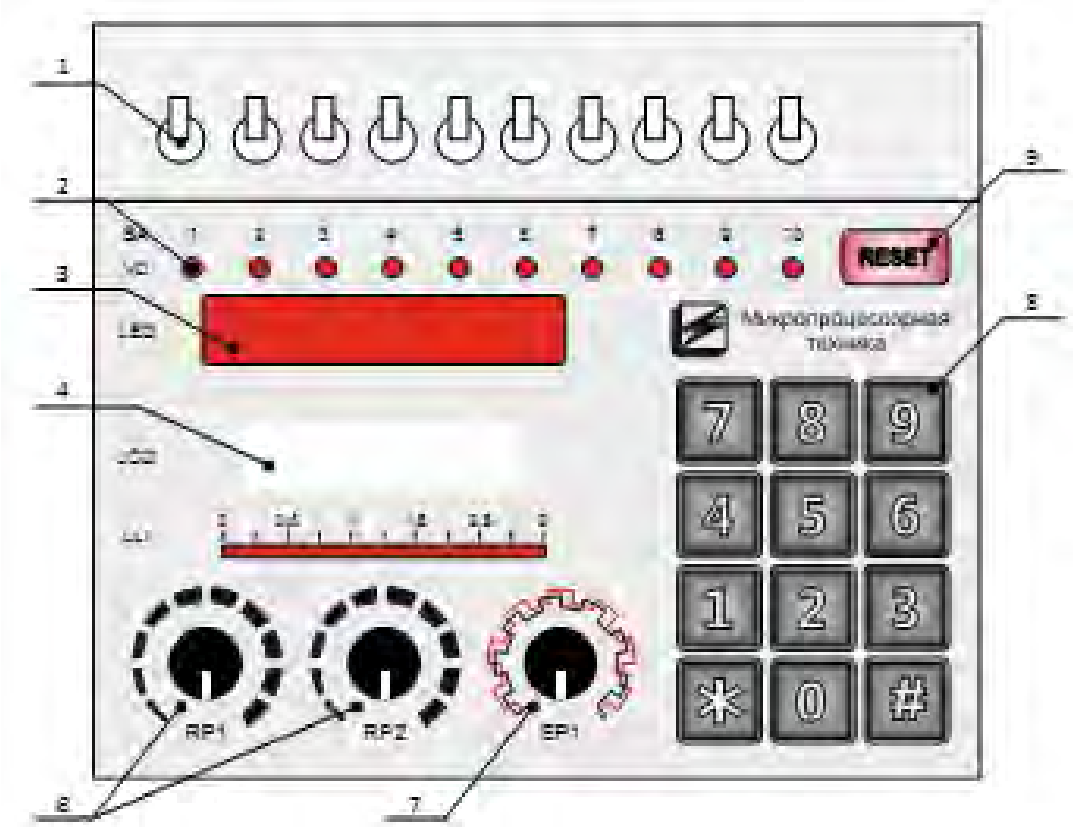



Рис. 1.3. Лицевая панель стенда

Задняя панель стенда представлена на рисунке 1.4. На задней панели стенда расположены:

1 – тумблер включения питания стенда;
2 – разъём для подключения внешних устройств; 3 – разъём для организации связи по последовательному интерфейсу;
4 – разъём USB-B для подключения стенда к компьютеру для программирования микроконтроллера.

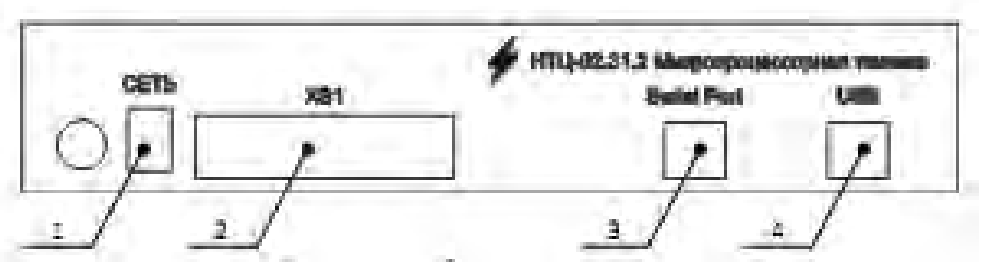



Рис. 1.4. Задняя панель стенда

## 2 Порядок работы учебного стенда

1) Подключить стенд к сети. Включить тумблер «Сеть» на задней панели стенда.
2) С помощью шнура USB AM-BM через гнездо «USB» на задней панели стенда подключить стенд к USB-порту компьютера.
3) Запустить MPLAB IDE.
4) С помощью пункта меню «File → Import…» открыть файл тестовой прошивки стенда «Test.hex».

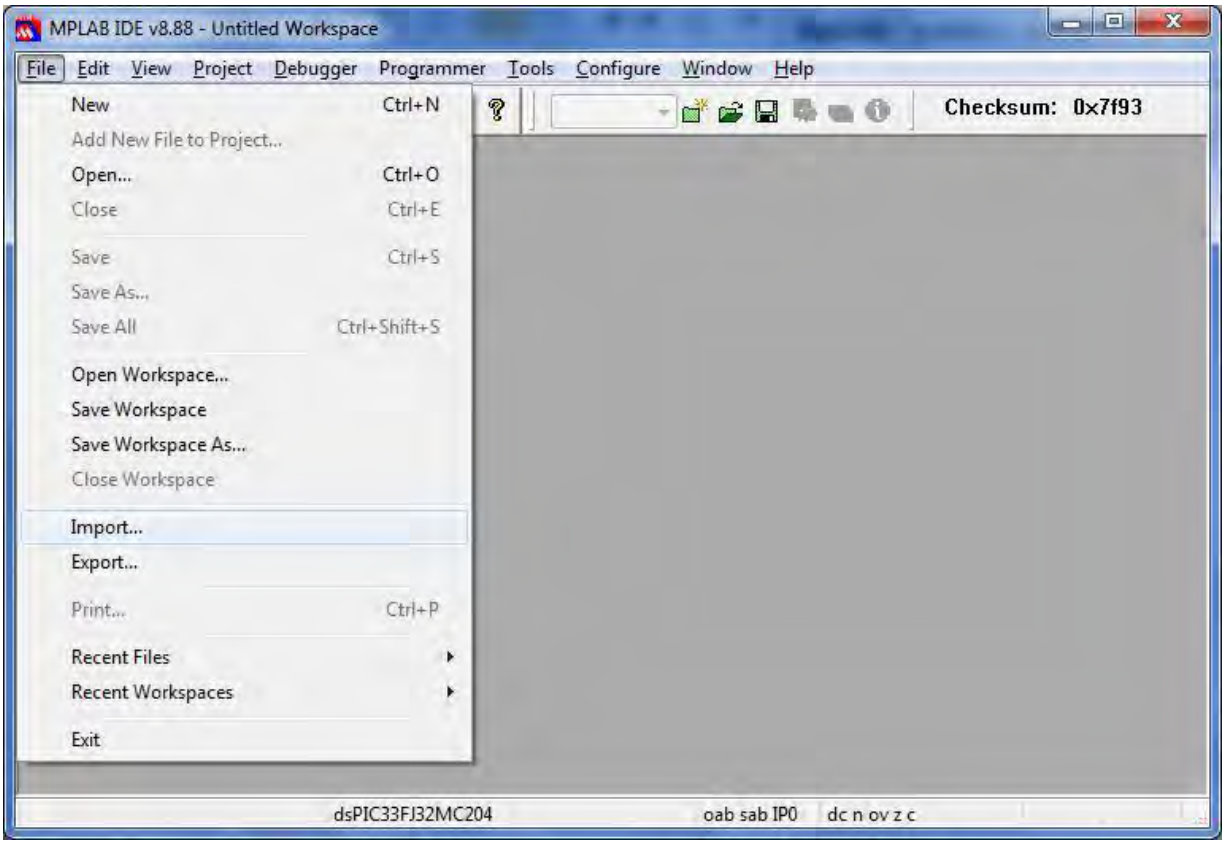



Рисунок 2.1 – Открытие файла тестовой прошивки стенда

- С помощью пункта меню «Programmer → Select Programmer» выбрать программатор PICkit 2 (рис 2.2).

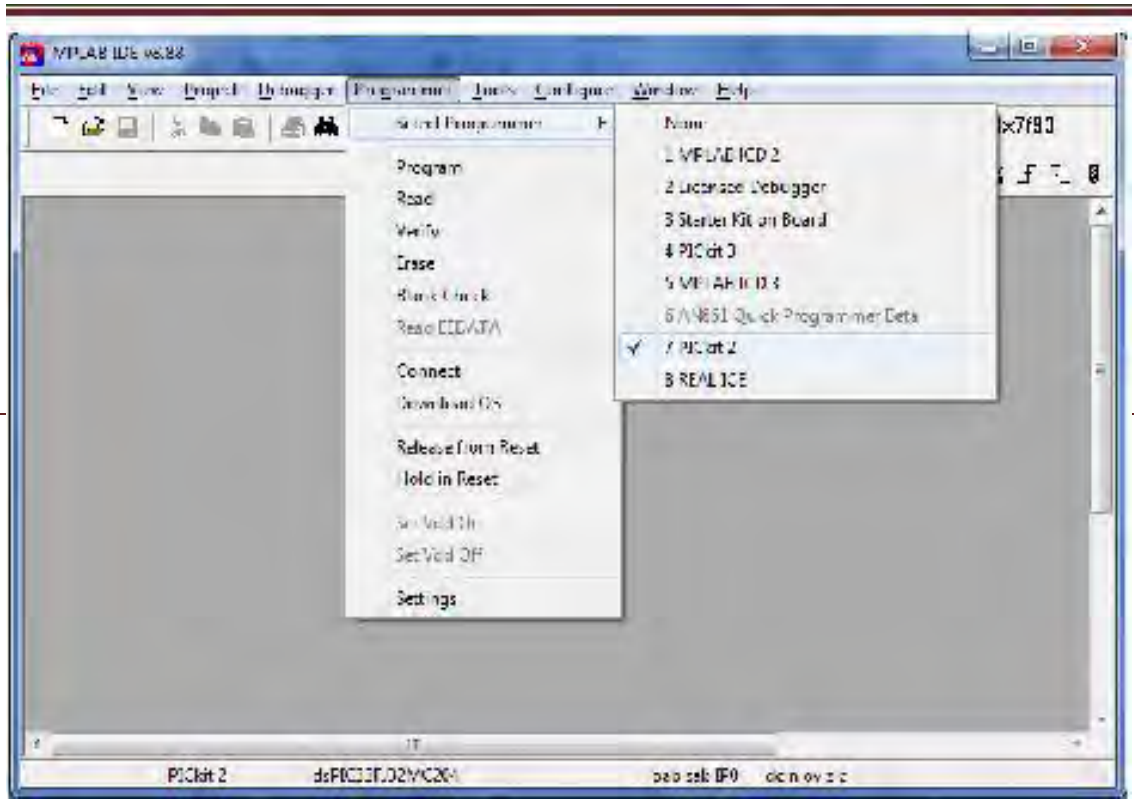



Рис. 2.2. Выбор программатора PICkit 2

1) Запрограммировать стенд тестовой прошивкой, воспользовавшись пунктом меню «Programmer → Program» (рис. 2.3).

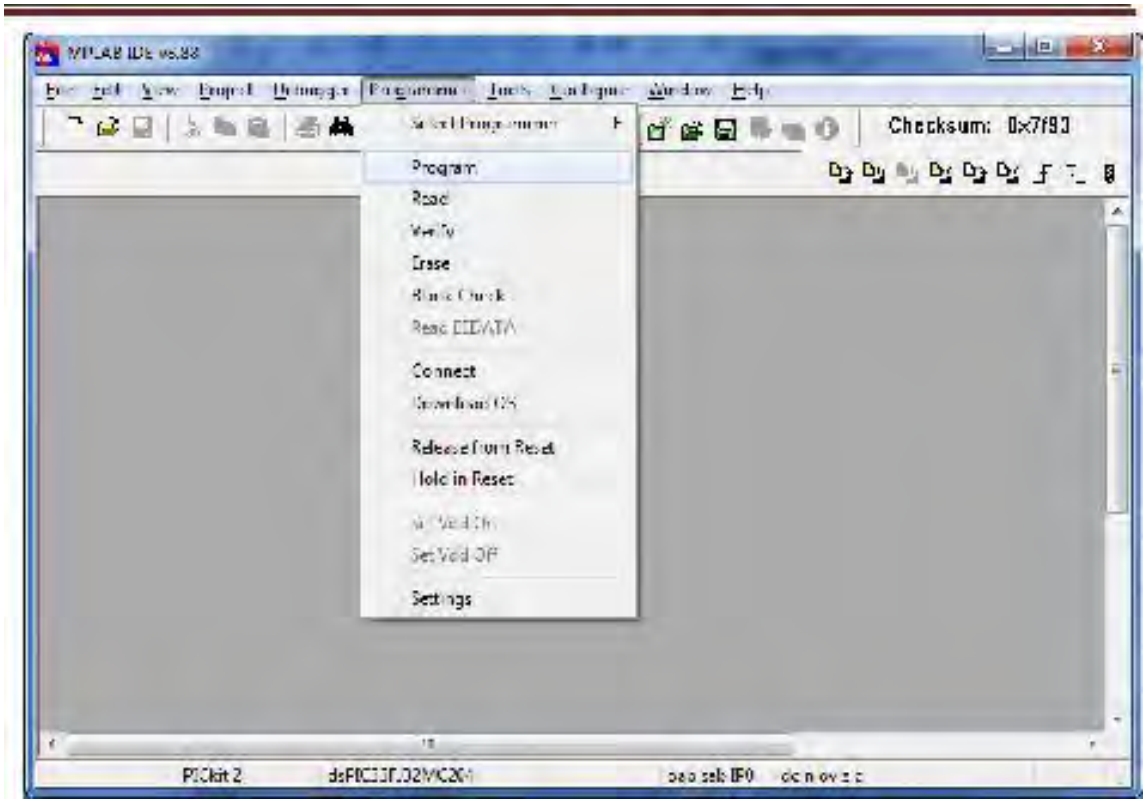



Рис. 2.3. Программирование стенда тестовой прошивкой

- Перевести микроконтроллер в рабочий режим, воспользовавшись пунктом меню «Programmer → Release from Reset».
- Закрыть MPLAB IDE.
- Изучить назначение органов управления стенда.
- По результатам выполненных операций сделать вывод.

## 3 Контрольные вопросы

1) Перечислите основные характеристики микроконтроллеров семейства dsPIC33?
2) Какие структурные элементы входят в состав микроконтроллеров семейства dsPIC33?
3) Каковы основные структурные элементы учебного стенда НТЦ-02.31.2?
4) Опишите органы управления лабораторного стенда.
5) Каков порядок работы со стендом?

## Лабораторная работа 2

## Изучение системы команд и основ работы микроконтроллера семейства dsPIC33

**Цель работы:**

Изучить состав и особенности выполнения команд микроконтроллера, программное обеспечение лабораторного стенда, освоить технику программной симуляции работы микроконтроллера на примере построения элементарной конструкции.

**Порядок выполнения работы:**

1) Изучить систему команд микроконтроллера.
2) Изучить программное обеспечение лабораторного стенда.
3) Составить программу в соответствии с индивидуальным заданием.
4) Написать исходный код и отладить программу в среде MPLAB IDE.
5) Исследовать выполнение программы в симуляторе MPLAB SIM.
6) Оформить отчёт по лабораторной работе.
7) Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Архитектура

Ядро dsPIC построено по модифицированной Гарвардской архитектуре с расширенной системой команд. Микроконтроллеры dsPIC33 поддерживают выполнение специфических для алгоритмов цифровой обработки сигналов инструкций (умножение с накоплением), специальные методы адресации (модульная, бит-реверсивная), знаковые вычисления с целыми числами и числами с фиксированной точкой.

### 1.2 Память программ

Карта памяти программ контроллеров dsPIC линейная и несегментированная. Счётчик команд (PC – Program Counter) имеет разрядность 23 бита, однако младший бит всегда равен 0 для обеспечения выравнивания данных при выборке инструкции. Следовательно, для задания непосредственно адреса используется 22 бита, и таким образом эффективное количество адресуемых инструкций равно $2^{22} \sim 4$ млн.

Адрес памяти более удобно рассматривать в виде младшего и старшего слова, при этом старший байт старшего слова не используется и равен 0. Младшее слово всегда располагается по четному адресу, а старшее – по нечетному. Адреса памяти программ всегда выравниваются по границе слова и при выполнении программного кода всегда инкрементируются и декрементируются на 2.

Карта программной памяти микроконтроллера dsPIC33fj32mc204 изображена на рис. 1.1.

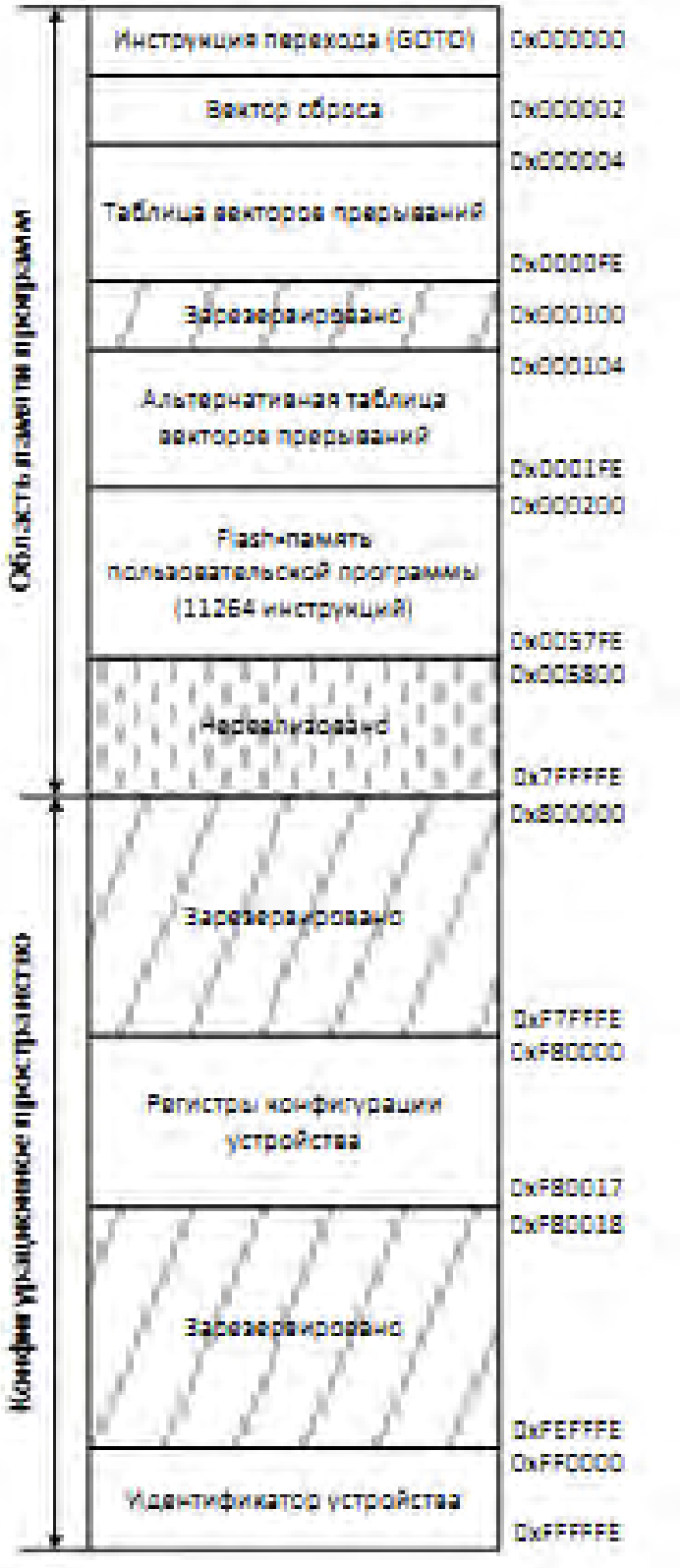



Рис. 1.1. Карта памяти программ микроконтроллера dsPIC33

Семейства dsPIC33 не имеют EEPROM, что связано с изменением технологии изготовления кристаллов. Поэтому при необходимости наличия в системе энергонезависимой памяти малого объема с большим количеством циклов перепрограммирования рекомендуется использовать внешние микросхемы памяти EEPROM с последовательным интерфейсом.

Пользовательским программам доступна область памяти в диапазоне адресов 0x000000…0x7FFFFF. Память программ организована в виде блоков, адресуемых пословно. Хотя адрес памяти является 24 битным, более удобно рассматривать любой адрес в виде младшего и старшего слова, при этом старший байт старшего слова не используется и равен 0. Младшее слово всегда располагается по четному адресу, а старшее — по нечетному.

По адресам 0x000000 и 0x000002 размещается команда перехода к фактическому началу программы, при этом по первому адресу располагается код операции инструкции goto, а по второму — собственно адрес точки входа в программу. Также в памяти программ размещаются две таблицы векторов прерываний. Одна из них располагается в диапазоне адресов 0x000004…0x0000FE, а вторая (альтернативная) — в диапазоне адресов 0x000104…0x0001FE.

Половину карты программной памяти занимает так называемое «конфигурационное пространство». Во всех 16-битных микроконтроллерах физически реализована только небольшая часть этого сектора – это слова конфигурации, определяющие режим работы контроллера после сброса и идентификационный код кристалла, предназначенный для определения контроллера аппаратными средствами разработки.

Физически программная память во всех контроллерах 16-битного семейства реализована в виде перепрограммируемой Flash-памяти. Все контроллеры поддерживают внутрисхемное программирование и программирование в ходе выполнения программы.

### 1.3 Память данных

Шина адреса данных микроконтроллеров dsPIC33 является 16-битной и позволяет адресовать до 64 кБ памяти ОЗУ, которая физически выполнена в качестве статической памяти с возможностью байтового доступа. Почти все инструкции, работающие с ОЗУ, имеют спецификатор, указывающий, будет ли осуществляться доступ к слову (16 бит) или к байту (8 бит). Карта памяти данных микроконтроллера dsPIC33fj32mc204 приведена на рис. 1.2.

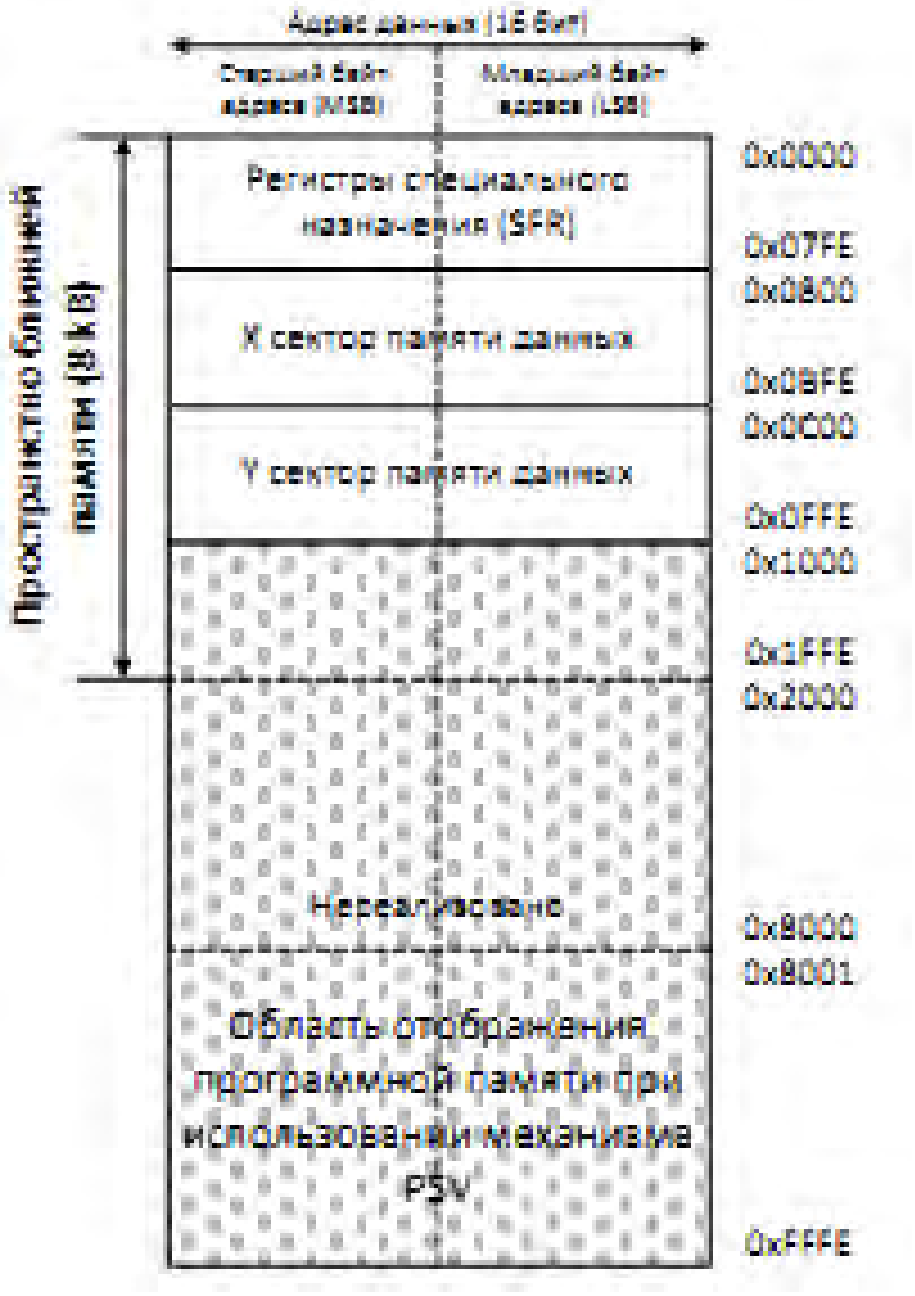



Рис. 1.2. Карта памяти данных микроконтроллера dsPIC33

1) начале ОЗУ расположена область регистров специального назначения (SFR) объемом 2 кБайта. Внутри семейства все регистры SFR расположены статически по одним адресам.

Данные в памяти выравниваются по границе слова, при этом младшие байты имеют четные, а старшие — нечетные адреса.

С адреса 0x0800 начинается сектор ОЗУ общего назначения, максимальный объем которого составляет 30 кБайт; в разных контроллерах только часть этого сектора может быть реализована физически. Верхнюю половину ОЗУ занимает область, в которую отображается часть программной

памяти при использовании механизма PSV.

2) семействе dsPIC33 реализовано два адресных генератора, что позволяет инструкциям DSP-ядра делать две выборки из ОЗУ за один командный такт. Это объясняет разделение области ОЗУ общего назначения на два сектора X и Y. Однако это разделение не является сегментированием или ограничением – для всех инструкций CPU-ядра сектор ОЗУ общего назначения линеен.

Первые 8 кБайт ОЗУ (включая область SFR) называются пространством ближней памяти (Near Data Space). Прямая адресация возможна только к этому сегменту. Остальная часть ОЗУ может быть адресована косвенно.

### 1.4 АЛУ, умножитель, поддержка деления

16-битное арифметико-логическое устройство (АЛУ) позволяет выполнять за один командный такт следующие операции: сложение, вычитание, битовый сдвиг и поразрядные логические операции, включая инверсию. Результаты операций влияют на статусные флаги. АЛУ выполняет операции с 16-битными словами или байтами в зависимости от синтаксиса инструкции.

В состав ядра входит умножитель 17 x 17 бит, который позволяет выполнять операции умножения 16 x 16, 16 x 8 и 8 x 8 бит, знаковые, беззнаковые и смешанные за один командный такт. При знаковом и смешанном умножении используется расширение знака. Операции умножения не влияют на статусные флаги. В качестве операндов используются рабочие регистры, результат умножения возвращается в указанную регистровую пару.

Аппаратная поддержка деления (итерационный метод) позволяет значительно сократить время выполнения математических алгоритмов. Операции деления 32/16 и 16/16 бит (знаковые и беззнаковые) выполняются за 18 командных тактов. В качестве операндов используются рабочие регистры, после выполнения деления доступен результат и остаток.

Ядро содержит модуль сдвига (barrel shifter), который позволяет выполнять за один командный такт операции сдвига и вращения на произвольное (до 15) количество бит. Сдвиг вправо может быть логическим (без расширения знака) или арифметическим (с расширением знака).

### 1.5 Система тактирования микроконтроллера

Микроконтроллер состоит из множества различных модулей, работу которых необходимо синхронизировать и тактировать. Для этих целей служит тактовый генератор. Тактовый генератор непрерывно формирует импульсы заданной частоты. Главной характеристикой генератора является частота.

Семейства dsPIC33 имеют переработанную схему конвейера – команда выполняется за 2 такта генератора. Таким образом, частота выполнения

команд соответствует частоте $F_{CY}$ и по умолчанию $F_{CY} = F_{OSC} / 2$. Максимальная производительность dsPIC33 составляет 40 MIPS (при тактовой частоте 80 MHz).

Микроконтроллер семейства dsPIC33 имеет возможность тактирования от одного из двух внутренних и двух внешних источников, позволяя также производить их настройку и переключение. Внутренние генераторы позволяют уменьшить количество деталей и размеры платы и применяются в тех случаях, когда не требуется высокая точность их работы. Внешние генераторы применяются при необходимости точной синхронизации нескольких устройств в схеме. На рисунке 1.3 изображена блок-схема системы тактирования микроконтроллера.

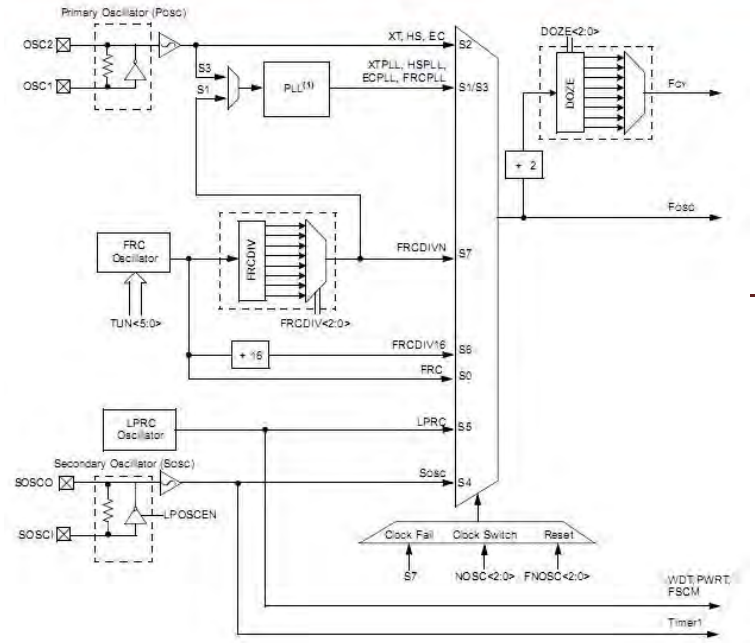



Рис. 1.3. Блок-схема системы тактирования микроконтроллера

Таким образом, схема тактирования имеет четыре источника – первичный кварцевый генератор (4-10 МГц), вторичный кварцевый генератор (32 кГц), внутренний RC генератор 7.37 МГц и внутренний низкочастотный RC генератор 512 кГц. Первичный генератор может тактировать схему умножения частоты с ФАПЧ (PLL), коэффициент умножения является дробным и настраивается программно. PLL-умножитель не может быть подключен к внутренним RC генераторам или ко вторичному кварцевому генератору. Микроконтроллер может тактироваться от внешнего источника частоты (0-40 МГц). Система тактирования содержит детекторы стабильности тактовой частоты - возможно автоматическое переключение на вторичный источник при сбое первичного.

Частота внутреннего генератора может программно корректироваться в пределах +/- 12% с шагом в десятые доли процента.

Для работы с тактовым генератором предназначены нижеперечисленные регистры:

1) FOSCSEL – регистр выбора генератора;
2) FOSC – регистр конфигурации генератора;
3) OSCCON – регистр управления генератором;
4) CLKDIV – регистр делителя частоты;
1) PLLFBD – регистр делителя частоты обратной связи PLL;
2) OSCTUN – регистр для настройки генератора FRC.

Регистры FOSCSEL и FOSC относятся к регистрам битов конфигурации.

## 1.6 Биты конфигурации

Микроконтроллеры семейства dsPIC33 имеют т.н. биты конфигурации, задающие такие параметры работы микроконтроллера, как источник тактирования, параметры защиты от нестабильности источника питания, параметры защиты кода программы и др. Такие биты программируются и устанавливаются единожды перед записью прошивки микроконтроллера и не могут быть изменены программно.

Установка битов конфигурации производится либо специальным инструментом среды программирования MPLAB IDE, либо специальной директивой в коде программы перед основной функцией программы.

Биты конфигурации микроконтроллера семейства dsPIC33 объединены в 4 регистра:

1) FOSC – регистр конфигурации тактового генератора;
2) FWDT – регистр конфигурации сторожевого таймера;
3) FBORPOR – регистр конфигурации контроля напряжения питания;
4) FGS – регистр конфигурации сегмента кода.

### a. Программная модель микроконтроллера

a. качестве рабочих используются 16 регистров процессора (обозначаемых W0..W15), которые могут функционировать как регистры данных или адресов. Все регистры ортогональны с точки зрения системы команд, то есть могут быть использованы

2) качестве операнда инструкции. Некоторые инструкции могут использовать в качестве операндов или регистров сохранения результата только определенный регистр или регистровую пару. Функция, выполняемая регистром, определяется режимом адресации, который используется в данной инструкции.

a. регистре W15 содержится указатель стека (Stack Pointer – SP), который

автоматически модифицируется при обработке исключений, вызовов подпрограмм и прерываний. Тем не менее, регистр W15 можно использовать в инструкциях в качестве обычного рабочего регистра. Во многих случаях такое использование упрощает манипуляции с указателем стека.

При инициализации микроконтроллера в регистр W15 записывается значение 0x0800, и это обеспечивает установку указателя стека на начало

области памяти, которая доступна пользователю. Указатель стека всегда декрементируется перед извлечением значения из стека и инкрементируется после помещения значения в стек.

Процессор микроконтроллеров dsPIC33 имеет 16 битный регистр состояния (Status Register – SR). Младшая часть регистра состояния (SRL) содержит флаги арифметических операций (нуля, переноса, переполнения и знака), а также 3 бита приоритета (IPL2…IPL0), в которых устанавливается текущий приоритет прерываний процессора. Биты регистра, а также их расшифровка представлены в таблицах 1.1 и 1.2 соответственно.

Таблица 1.1

Биты регистра SR

| *Обозначение* | OA | OB | SA | SB | OAB | SAB | DA | DC |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | IPL<2:0> | | | RA | N | OV | Z | C |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.2

Значение некоторых битов регистра SR

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 7-5 | IPL<2:0> | Приоритет прерываний процессора |
| 3 | N | Бит отрицательности АЛУ<br>N = 1: результат вычисления был отрицательный<br>N = 0: результат вычисления был неотрицательный |
| 2 | OV | Бит переполнения АЛУ<br>Бит используется для вычислений со знаковыми числами.<br>Индицирует, было ли переполнение, которое привело к тому,<br>что бит знака изменил состояние.<br>OV = 1: переполнение произошло<br>OV = 0: переполнение не произошло |
| 1 | Z | Бит ноля АЛУ<br>Z = 1: действие, влияющее на бит ноля, было выполнено<br>Z = 0: последнее действие, влияющее на бит ноля, привело к его<br>Сбросу |
| 0 | C | Бит переноса АЛУ<br>C = 1: произошёл перенос старшего бита результата<br>C = 0: переноса старшего бита результата не произошло |

### 1.8 Режимы адресации

Инструкции процессора dsPIC33 могут быть регистровыми и адресными. Основное различие между этими типами инструкций состоит в том, что регистровые инструкции модифицируют содержимое регистров, в то время как адресные инструкции используют содержимое рабочих регистров для доступа к данным, находящимся в памяти. В свою очередь адресные инструкции могут быть как с прямой регистровой адресацией, так и с косвенной регистровой адресацией.

Прямая регистровая адресация используется для доступа к содержимому 16 рабочих регистров W0…W15, причем обращение возможно как к целому слову (16 бит), так и к байту.

Косвенная регистровая адресация применяется для доступа к любой ячейке памяти данных посредством формирования исполнительного адреса данных по содержимому одного или нескольких рабочих регистров. Таким образом, содержимое рабочего регистра или регистров является указателем на область памяти данных. Дополнительные возможности такого метода адресации обеспечиваются механизмами пред- и пост- инкремента содержимого регистра, что позволяет последовательно обрабатывать непрерывный диапазон адресов в памяти данных.

Примеры применения различных методов адресации представлены ниже:

1) прямая адресация, адрес операнда указывается непосредственно в операционном слове инструкции. Операнд должен быть расположен в области Near Data Space (первые 8 кБайт ОЗУ):

```
add  0x900, W0     ;Сложение значения по адресу 0x900 с W0.
                   ;Результат сохраняется в W0
```

2) расширенный режим инструкции mov – прямая адресация, но может адресоваться вся память ОЗУ

```
mov 0x2500, W7     ;Сохранение значения по адресу 0x2500 в
регистр W7
```

1) прямая адресация регистров. В качестве операндов используются значения рабочих регистров:

```
ior W0, W2, W5 ;Поразрядное логическое ИЛИ регистров W0
                  и W2, ;сохранение результата в W5
```

2) косвенная адресация – может использоваться в большинстве инструкций:

```
add  W4, [W5], [W6]      ;Сложение W4 со словом по указателю
W5,
                         ;сохранение результата по указателю W6
```

1) косвенная адресация с пре/пост инкрементом и декрементом, при этом учитываются правила адресной арифметики в зависимости от типа операнда (байт или слово):

```
;Увеличение указателя w0 на 2, перемещение
значения ;по указателю W0 в ячейку, на которую
указывает W1, ;уменьшение указателя w1 на 2.
mov [++W0], [W1--]
```

1. косвенная адресация со смещением:

```
;Получение указателя на ячейку операнда, путем сложения
W4 и W5, ;перемещение значения по полученному
указателю по адресу W6, ;увеличение значения W6 на 2
mov [W4 + W5], [W6++]
```

Поддержка различных методов адресации позволяет получать очень компактный код при использовании компиляторов с языков высокого уровня. Следует так же отметить возможность вызова и перехода по значению в регистре (аналог вызова функции по указателю в Си) и инструкцию запрещения прерываний на определенное количество командных тактов (позволяет выполнять набор инструкций как одну атомарную).

## 1.9 Система команд

16-битные микроконтроллеры Microchip имеют расширенный набор инструкций, большинство из которых поддерживает операции типа «чтение-модификация-запись», что позволяет работать с данными напрямую в ОЗУ, не используя рабочие регистры.

Семейство dsPIC33 имеет 83 инструкции (включая инструкции DSP-ядра). Почти все инструкции выполняются за один командный такт. В зависимости от типа операндов, инструкции могут быть слово-, байт- либо бит-ориентированными. Большинство слово- и байт-ориентированных инструкций являются трехоперандными (типа A = B + C). Большинство бит-ориентированных инструкций имеют два операнда.

Систему команд микроконтроллеров dsPIC33 можно разделить на несколько групп:

- команды перемещения;
- команды математических операций;
- команды логических операций;
- команды сдвига и циклического сдвига;
- команды работы с битами;
- команды сравнения/выбора;
- команды условных переходов;
- команды работы со стеком;
- команды управления.

**Команды перемещения**

В командах перемещения, большинство из которых имеет мнемоническое обозначение mov, используется хотя бы один регистр. Регистровые инструкции могут использовать W регистр как регистр данных или регистр, в который помещается смещение адреса. Пример использования команды:

```
mov #0x19C7,
W0 mov W0,
W1
```

В первой инструкции константа 0x19C7 командой mov помещается в регистр W0. Затем содержимое регистра W0 пересылается в регистр W1. Таким образом, после выполнения второй инструкции в регистре W1 будет содержаться значение 0x19C7.

```
mov #0x1234,
W0 mov W0,
[W1]
```

В данном примере первая команда mov помещает в регистр W0 константу 0x1234. Вторая команда помещает содержимое регистра W0 по адресу, находящемуся в регистре W1.

**Команды математических операций**

Микроконтроллеры семейства dsPIC33 обладают обширным набором инструкций для выполнения математических и логических действий над операндами. В АЛУ процессора предусмотрено выполнение 4 основных математических действий над целыми знаковыми и беззнаковыми числами: сложения, вычитания, умножения и деления. Для каждого из этих действий предусмотрен ряд инструкций, в которых используются различные типы адресации, делающие выполнение таких операций весьма эффективным.

К математическим инструкциям относятся следующие:

- add – сложение;
- sub – вычитание;
- inc – увеличение на 1;
- inc2 – увеличение на 2;
- dec – уменьшение на 1;
- dec2 – уменьшение на 2;
- mul – умножение;

- div.xx – деление, где xx – указывает тип и длину операндов.
  Далее приведено несколько примеров использования этих инструкций.

```
mov #34, W0
add W0, #7,
W1
```

В этой программе команда add имеет 3 операнда. К содержимому регистра W0 прибавляется 7, и результат записывается в регистр W1. После выполнения операции там будет находиться значение 41 (0x29).

```
mov #7, W1
add W1, #5, [W0]
```

В данном случае команда add складывает содержимое регистра W1 (0x7) с константой 5 и помещает результат по адресу, находящемуся в регистре W0. После выполнения операции в этой ячейке будет находиться значение 12 (0xC).

Группа математических команд содержит инструкции, позволяющие работать с многобайтовыми данными. Эти команды при выполнении операций учитывают флаг переноса, который мог быть установлен как следствие переноса или заема в результате операции с предыдущими словами. К таким инструкциям относятся addc и subb.

**Команды логических операций**

Эти команды позволяют выполнять булевые одноместные и двуместные операции. К таким операциям относятся хорошо известные двуместные операции логическое «И», логическое «ИЛИ», исключающее «ИЛИ» и одноместная операция отрицания «НЕ». В эту группу команд включены инструкции очистки (обнуления) операнда и установки, когда все биты операнда устанавливаются в 1.

Инструкции, выполняющие операцию логического «И», имеют мнемоническое обозначение *and*, логического «ИЛИ» — *ior*, исключающего «ИЛИ» — *xor*. Операция отрицания «НЕ» реализована посредством инструкции *com*. Кроме того, в эту группу команд включена команда *neg*, позволяющая инвертировать знак числа. Инструкции логических операций весьма полезны при выделении и анализе отдельных битов операндов.

Применение этих команд показано в следующих примерах.

```
mov    #0xFC,
W0          mov
#0x13,      W1
```

```
and W0, W1,
W2
```

Здесь выполняется операция логического «И» над операндами, помещенными в регистры W0 и W1. Команда and помещает результат операции в регистр W2. При указанных значениях операндов W0 и W1 итоговое значение в регистре W2 будет равно 0x10. Если инструкцию «and W0, W1, W2» заменить инструкцией «ior W0, W1, W2» (логическое «ИЛИ»), то итоговое значение в регистре W2 станет равным 0xFF. При использовании в программе команды xor (исключающее «ИЛИ») в регистре W2 после выполнения операции будет содержаться значение 0xEF.

```
mov #0xC,
W0 com
W0, W1
```

В этом примере показано применение инструкции com. Эта инструкция инвертирует все биты исходного операнда. В данном случае инвертируется содержимое регистра W0 (0x000C), а результат, который будет равен 0xFFF3, помещается в W1.

```
mov #0xFFF4,
W0 neg W0,
W1
```

Программа инвертирует знак числа при помощи команды neg. Команда neg использует стандартный алгоритм получения инверсии числа: сначала все биты операнда инвертируются, затем к младшему биту прибавляется 1. В данном примере при исходном значении операнда в регистре W0, равном 0xFFF4 (дополнительный код отрицательного числа –12), результат выполнения команды neg, помещенный в регистр W1, будет равен 0x000C, что соответствует положительному числу 12.

**Команды сдвига и циклического сдвига**

Эта группа команд часто используется для преобразования последовательного потока битов в параллельный двоичный код и обратно, а также для операций быстрого целочисленного умножения и деления на степень двойки.

К командам сдвига и циклического сдвига относятся следующие:

- asr – арифметический сдвиг вправо;
- lsr – логический сдвиг вправо;
- rlc – сдвиг влево через флаг переноса;
- rlnc – сдвиг влево без использования флага переноса;

- rrc – сдвиг вправо через флаг переноса;
- rrnc – сдвиг вправо без использования флага переноса;
- sl – сдвиг влево.

Пример применения инструкции сдвига:
```
mov #34, W0
lsr W0, #2, W1
```

В данном примере выполняется сдвиг содержимого регистра W0, которое равно 34 (0x22), на 2 позиции вправо, а результат сохраняется в регистре W1. Таким образом, после выполнения команды lsr W0, #2, W1 в регистре W1 будет содержаться значение 8 (0x8).

**Команды работы с битами**

Команды работы с битами, как следует из названия, предназначены для манипуляций с отдельными битами операндов. Такая возможность является весьма востребованной, поскольку в большинстве случаев системы на базе микроконтроллеров должны работать с отдельными сигнальными битами внешних устройств. Битовые команды позволяют анализировать состояние отдельных битов, что дает возможность организовать ветвления и переходы в программе, а также устанавливать или сбрасывать отдельные биты операнда.

К командам работы с битами относятся следующие:

- bclr – сброс бита;
- bset – установка бита;
- btg – инвертирование бита;
- btst – проверка бита.

Пример применения команд работы с битами:
```
mov #34,
W0 bset
W0, #0 bclr
W0, #5
```

В данном примере в регистр W0 записывается значение 34 (0x22). Затем команда «bset W0, #0» устанавливает бит 0 этой переменной, в результате чего содержимое регистра становится равным 35 (0x23). Следующая за ней инструкция bclr W0, #5 сбрасывает бит 5 регистра, в результате чего окончательное его значение будет равно 3 (0x3).

Для быстрого анализа состояния отдельных битов переменной в систему команд микроконтроллеров dsPIC33 включены инструкции *fbcl*, *ff1l* и *ff1r*. Эти инструкции очень полезны в системах сбора данных при сканировании,

например, сигнальных выводов периферийных устройств. Краткое описание этих инструкций:

2.1 fbcl — определяет изменение состояния битов от младшего к старшему (слева направо);

2.2 ff1l — находит первый ненулевой бит при проходе слева направо;

2.3 ff1r — находит первый ненулевой бит при проходе справа налево.

Пример использования инструкции ff1r:

```
mov #8,
W2 ff1r
W2, W3
```

В примере в регистр W2 заносится значение 8. Инструкция ff1r обнаруживает первый единичный бит в регистре W2 на позиции 4 (отсчет ведется от 1) и записывает это значение в регистр W3. Таким образом, регистр W3 будет содержать значение 4.

Среди команд битовых операций инструкции установки (*bset*), сброса (*bclr*) и переключения (*btg*) битов чаще всего применяются для управления внешними исполнительными устройствами, подключенными к портам вывода микроконтроллера. Пример использования команды btg для переключения состояния бита 0 порта PARTA:

```
btg PORTA, #0
```

**Команды сравнения/выбора и условных/безусловных переходов**

К группе команд сравнения/выбора относятся команды вида *BTxx* и *CPxx*, где аббревиатура BT означает «Bit Test», CP означает «Compare», а символы *xx* — одно из условий. О функциях команд понятно по их названию. Команды BTxx проверяют состояние указанного бита (0 или 1) и по результатам проверки выполняют определенное действие. Команды CPxx выполняют сравнение значений операндов, в результате чего устанавливаются флаги в регистре состояния SR процессора. Дальнейшие действия выполняются с помощью команд условных переходов bra xx, где xx — условие выполнения перехода.

Команды группы условных/безусловных переходов предназначены для управления ходом вычислительного процесса и осуществляют переход к различным секциям программы по результатам проверки флагов состояния процессора. В предыдущих примерах программного кода мы уже использовали некоторые из этих команд, сейчас же остановимся на них более подробно.

Наиболее широко при разработке программного обеспечения

используются инструкции *bra xx* (xx — код условия), *call* и *goto*.

Команда *bra xx* осуществляет переход по адресу программы, содержащемуся в теле инструкции, при выполнении условия *xx*. В общей сложности эта команда может проверять выполнение следующих условий:

2.5 C — установлен флаг переноса;
2.6 GE — больше или равно;
2.7 GEU — больше или равно для операций без знака;
2.8 GT — больше;
2.9 GTU — больше для операций без знака;
2.10 LE — меньше или равно;
2.11 LEU — меньше или равно для операций без знака;
2.12 LT — меньше;
2.13 LTU — меньше для операций без знака;
2.14 N — результат предыдущей операции отрицательный;
2.15 NC — перенос (заем) отсутствует;
2.16 NN — результат предыдущей операции неотрицательный;
2.17 NOV — переполнения нет;
2.18 NZ — результат предыдущей операции ненулевой;
2.19 OV — возникло переполнение;
2.20 Z — установлен бит нуля в регистре состояния.

Например, команда «bra z, label1» выполнит переход на метку label1 программы, если установлен бит Z в регистре состояния процессора.

Кроме того, команда bra имеет две модификации, позволяющие выполнять безусловный переход:

2.6 *bra метка*, где метка должна находиться в пределах от –32768 до +32767 слов памяти программ относительно команды bra;
2.7 *bra Wn*, где Wn — рабочий регистр.

К командам переходов относятся и команды вызова процедур call и безусловного перехода goto. Инструкции call и goto в качестве операнда могут использовать либо метку, либо один из рабочих регистров. Если в качестве адреса перехода использовать содержимое какого-либо из рабочих регистров, то с помощью одной инструкции bra или call можно организовать несколько ветвлений в программе, например, по принципу оператора switch…case языка Си.

Пример использования команд:

```
        mov #0x1, W0
        mov #handle(Sub1),
        W2 btss W0,#0
        mov #handle(Sub2),
        W2 call W2
Sub1:
        …
```

```
        return
Sub2:
        …
        return
```

Программа работает следующим образом: с помощью команды btss W0, #0 анализируется бит 0 регистра W0: если он равен 0, то выполняется следующая за инструкцией команда, а если он равен 1, то следующая за командой инструкция пропускается. Таким образом, если бит 0 установлен, то в регистре W2 остаётся адрес вызова процедура Sub1, если же бит 0 сброшен, то в регистр W2 перезаписывается адрес вызова процедуры Sub2. Для того чтобы вызвать процедуру посредством адреса, содержащегося в рабочем регистре W2, выполняется команда «call W2». В данном примере бит 0 регистра W0 принудительно устанавливается в 1, поэтому вызывается процедура Sub1.

В примере значение стартового адреса одной из процедур (Sub1 или Sub2) записывается в регистр с помощью специального оператора handle в одной из команд:

```
mov
#handle(Sub1),W2
mov
#handle(Sub2),W2
```

**Команды работы со стеком**

Программный стек управляется командами *push* и *pop*. Эти инструкции эквивалентны инструкции mov с регистром W15 в качестве указателя приёмника. Для сохранения данных в стек необходимо выполнить следующую команду:

```
push W0
```

Значение, содержащееся на вершине стека, может быть получено следующей инструкцией:

```
pop W0
```

Во время выполнения инструкции вызова подпрограммы call, значение счётчика команд (PC) сохраняется в стек. Благодаря этому, после завершения работы подпрограммы, основная программа может продолжить своё выполнение с правильной позиции. Сохранение счётчика команд в стек происходит следующим образом: биты 15..0 располагаются по первому доступному адресу в стеке, а биты 22..16 помещаются по следующему адресу, при этом старшие биты второго слова дополняются нолями.

## 2 Пример выполнения работы

*Задача:* Вычислить значение выражения (25 + 13) * (18 – 9), результаты промежуточных вычислений хранить в регистрах. Выделить младшую тетраду результата и поместить её в старшую.

Листинг программы:

```
                                ; W0 =
mov         #25, W0             25
                                ; W1 =
mov         #13, W1             13
                                ; W2 =
add         W0, W1, W2          W0      + W1
                                ; W3 =
mov         #18, W3             18
mov         #9, W4              ; W4 = 9
                                ;    =
sub         W3, W4, W5          W5  W3  - W4
                                ;    =
mul.ss      W2, W5, W6          W6  W2  * W5
                                ;    =   &
and         #0x000F, W6         W6  W6  0x000F
                                ;    =
sl          W6, #12, W6         W6  W6  << 12
```

## 3 Варианты индивидуальных заданий к лабораторной работе

2.7 Загрузить в регистр число 15. Сложить его с 25 и результат поместить на вершину стека. Поместить по адресу 20h внутренней памяти данных младшую десятичную цифру результата, а по адресу 21h – старшую.

2.8 Найти разницу чисел 4836 и 2454. Младший байт результата поделить на 2. Поместить по адресу 30h внутренней памяти данных младшую десятичную цифру результата, а по адресу 32h – старшую.

2.9 Найти адрес ячейки памяти данных путем перемножения двух чисел 0Ch и 0Eh. В эту ячейку записать результат логической операции "исключающее или" между текущим содержимым регистра W0 и числа 09h.

2.10 Найти частное чисел 236 и 59. Результат умножить на 8 используя операции сдвига. По вычисленному таким образом адресу ячейки внутренней памяти данных размесить результат двойного декремента полученного числа.

Загрузить регистр W7 числом 023h. Найти сумму W7 + 32. В ячейку

внутренней памяти данных, расположенную по вычисленному таким образом адресу, загрузить число десятичных единиц результата сложения.

3.1 Вычислить значение выражения (81 + 64) * (112 - 25) OR 10011010b, сохраняя промежуточные результаты в стеке.

3.2 Найти разницу чисел 4801 и 209. Число десятичных единиц старшего байта результата поместить в старшую тетраду порта RA. Младшую тетраду оставить без изменений.

## 4 Контрольные вопросы

1. Сколько команд включает в себя система команд микроконтроллера?
2. На какие группы разделены команды микроконтроллера?
3. Как длительность машинного цикла микроконтроллера соотносится с его тактовой частотой?
4. Для чего предназначены биты конфигурации микроконтроллера и в чём их особенность?
5. Сколько источников тактирования может иметь микроконтроллер?
6. Каким образом настраивается источник тактирования микроконтроллера?
7. Какую функцию выполняет система прерываний?
8. Какое событие может быть причиной возникновения прерывания?

## Лабораторная работа 3

## Изучение системы программирования микроконтроллеров dsPIC33 с помощью языка программирования ассемблер

**Цель работы**:

Изучить алгоритмы типовых программ обработки информации, основные правила программирования и наиболее употребительные директивы. Составить алгоритм и программу обработки, исследовать ход её выполнения.

**Порядок выполнения работы:**

### 1 Краткие теоретические сведения

### 1.1 Обзор MPLAB ASM30 ассемблера

Язык ассемблера – машинно-ориентированный язык низкого уровня с командами, соответствующими командам микроконтроллера.

Условно текст программы на ассемблере можно разбить на следующие секции:

1. блок определений
   - o секция подключаемых файлов;
     секция определения битов конфигурации;
     секция определения констант;
     секция определения макросов;
     1.7секция объявления переменных.
- блок кода
  1.7обработчик прерываний;
  о основной цикл
  программы; о
  подпрограммы

Таким образом, в блоке определений указывается модель используемого контроллера, подключаются заголовочные файлы, объявляются константы и переменные, содержатся иные директивы, определяющие параметры работы ассемблера и варианты сборки программы. В блоке кода содержатся непосредственно исполняемые микроконтроллером инструкции, сгруппированные в подпрограммы и обработчики событий.

Пример исходного файла на языке ассемблер:

```
.list p=p33FJ32MC204
.include "p33FJ32MC204.inc"
.global __reset              ;Метка начала кода


__reset:
```

```
main:
        nop
        goto main

service:
        no
        p
        no
        p
        retfie
.end                            ;Конец кода
```

Основу языка ассемблера составляют директивы и инструкции. Директивы ассемблера интерпретируются во время выполнения работы ассемблера и используются для определения секций памяти, инициализации констант, декларирования и определения символов и т.д. Инструкции являются командами микроконтроллера, непосредственно исполняются им во время работы.

### 1.2 Общий формат инструкций и директив

Общий формат инструкций и директив ассемблера следующий:

[метка:] инструкция [операнды]
[;комментарии] [метка:] директива
[аргументы] [;комментарии]

Таким образом, каждая строка исходного файла может содержать до четырёх информационных полей:

- метка;
- мнемоника команды;
- операнды команды;
- комментарии.

Метки используются для отметки позиции в коде. Во время компоновки, метки определяют адреса в памяти. Метки должны начинаться с первой колонки. За меткой должно следовать двоеточие (:). Метка должна начинаться с символа латинского алфавита или символа подчёркивания (_) и может состоять из цифр и букв латинского алфавита и символа подчёркивания (_).

Мнемоники инструкций микроконтроллера, директивы ассемблера и макрокоманды должны начинаться со второй (и далее) колонки.

Операнды и аргументы следуют за мнемоникой команды. Операнды должны быть отделены от мнемоники не менее чем одним символом пробела

либо табуляции. Список операндов разделяется запятыми. Операнды используются в инструкциях для обеспечения информации об источнике и приёмнике. Аргументы подобны операндам и используются как источник и приёмник информации директив.

Любой текст до конца строки после символа «;» трактуется как комментарий. Комментарии могут следовать за операндами, мнемониками и метками и могут начинаться в любой колонке. Строковые константы, содержащие символ «;», как комментарий не воспринимается. Так же комментарии могут быть многострочными. В этом случае начало комментариев обозначается сочетанием символов «/*», а конец – сочетанием «*/».

### 1.3 Директивы ассемблера

Существует 5 основных типов директив:

- директивы контроля – управляют созданием разделов условно компилированного кода;
- директивы данных – управляют разделением памяти и назначением символических имён переменным и константам;
- директивы листинга – определяют формат и состав файла листинга. Эти директивы позволяют указывать заголовки, нумеровать страницы и настраивать другие параметры;
- макро директивы – управляют работой макросов и распределением данных в теле макроса;
- директивы объектного файла – используются только при создании объектного файла.

Основные директивы ассемблера перечислены ниже.

.list

Директива .list используется для управления процессом сборки программы. В частности, с помощью данной директивы можно указать используемый микроконтроллер, систему счисления по умолчанию, параметры работы со строками и т.д.. Пример использования директивы:

```
.list p=p33fj32mc204
```

.include

Директива .include добавляет содержимое указанного файла в исходный файл. Эффект аналогичен копированию полного текста включаемого файла в место расположения директивы. Параметр директивы – подключаемый файл – может указываться как с полным путём, так и без. Во втором случае поиск файла будет осуществляться в текущей рабочей директории, директории исходного файла и служебных директориях. Пример использования директивы:

```
.include “p33fj32mc204.inc”
.global
```

Директива .global используется для того, чтобы позволить меткам, определённым внутри файла, использоваться в другом файле. Пример использования директивы:

```
.global __reset
.global __OscillatorFail
.global __AddressError
```

В данном примере метки сделаны глобальными, чтобы компоновщик использовать их как адрес для перехода программы в указанные точки при наступлении соответствующих событий. Метка __reset используется для обозначения начала кода и используется как адрес для перехода из вектора сброса.

.section

Директива .section декларирует секцию памяти. Атрибутами, следующими за директивой, задается расположение секции – например, в памяти RAM либо в программной памяти. Пример использования директивы:

```
.section .data
.data
```

Директива используется для информирования ассемблера, что последующие данные будут помещены в секцию инициализированных данных. Если адрес секции не определён, то он будет назначен автоматически при связи объектных файлов.

.text

Директива .text используется для информирования ассемблера, что следующий код будет помещён в секцию программной памяти.

.equ

Директива .equ используется для определения символа и присвоения ему значения. Пример использования директивы:

```
.equ FCY, #7370000
```

В данном примере символу FCY присваивается литеральное значение 7370000. В таком контексте FCY является константой, которая может использоваться в коде.

.hword

Директива .hword используется для объявления инициализированных данных в пределах секции. Данные могут быть в виде констант, внутренних и внешних меток или их выражений. Пример использования директивы:

```
MinX: .hword 0x7FFF
```

.extern

Директива .extern используется для объявления переменной либо метки, которая может использоваться в данном файле кода, однако определена как глобальная в другом файле. Пример использования директивы:

```
.extern label
.end
```

Директива .end используется для обозначения окончания ассемблерного исходного файла. Пример использования директивы:

```
.list
p=p33fj32mc204
; текст
программы

.end
```

### 1.4 Числовые константы и системы счисления

MPASM поддерживает шестнадцатеричную, десятичную и двоичную системы счисления. Для обозначения числового значения используется символ «#».

Синтаксические правила числовых значений приведены в таблице 1.1.

Таблица 1.1

Синтаксис числовых значений

| *Тип* | *Синтаксис* | *Пример* |
|---|---|---|
| Шестнадцатеричный | 0x< числовое значение > | #0x9f |
| Десятичный | <числовое значение> | #123 |
| Двоичный | 0b<числовое значение> | #0b11 |

### 1.5 Текстовые строки

Текстовые строки могут состоять из знаков ASCII. Определение строки завершается специальным символом окончания строки – «\n».

## 2 Пример выполнения работы

*Задача:* Вычислить значение выражения (25 + 13) * (18 – 9), результаты промежуточных вычислений хранить в регистрах. Выделить младшую тетраду результата и поместить её в старшую. Проверить корректность выполнения алгоритма и результаты промежуточных вычислений в симуляторе. Указать результат выполнения алгоритма.

Листинг программы:

```
        .include "P33FJ32MC204.inc"
        .global __reset

__reset:
                                        ; W0 =
        mov     #25, W0                 25
                                        ; W1 =
        mov     #13, W1                 13
                                        ; W2 =
        add     W0, W1, W2              W0      + W1
                                        ; W3 =
        mov     #18, W3                 18
                                        ;
        mov     #9, W4                  W4  = 9
                                        ;   =
        sub     W3, W4, W5              W5  W3  - W4
                                        ;   =
        mul.ss  W2, W5, W6              W6  W2  * W5

        and     #0x000F, W6
                                        ;   =   <<
        sl      W6, #12, W6             W6  W6  12

        .end
```

## 3 Варианты индивидуальных заданий к лабораторной работе

Разработать программу в соответствии с вариантом задания. Проверить корректность выполнения алгоритма и результаты промежуточных вычислений в симуляторе. Пометить промежуточные значения переменных на каждом шаге и указать результат выполнения алгоритма.

- Загрузить в регистр число 15. Сложить его с 25 и результат поместить на вершину стека. Поместить по адресу 20h внутренней памяти данных младшую десятичную цифру результата, а по адресу 21h – старшую.
- Найти разницу чисел 4836 и 2454. Младший байт результата поделить на 2. Поместить по адресу 30h внутренней памяти данных младшую десятичную цифру результата, а по адресу 32h – старшую.
- Найти адрес ячейки памяти данных путем перемножения двух чисел 0Ch и 0Eh. В эту ячейку записать результат логической операции "исключающее или" между текущим содержимым регистра W0 и числа 09h.
- Найти частное чисел 236 и 59. Результат умножить на 8 используя операции сдвига. По вычисленному таким образом адресу ячейки внутренней памяти данных размесить результат двойного декремента полученного числа.
- Загрузить регистр W7 числом 023h. Найти сумму W7 + 32. В ячейку внутренней памяти данных, расположенную по вычисленному таким образом адресу, загрузить число десятичных единиц результата сложения.
- Вычислить значение выражения (81 + 64) * (112 - 25) OR 10011010b, сохраняя промежуточные результаты в стеке.
- Найти разницу чисел 4801 и 209. Число десятичных единиц старшего байта результата поместить в старшую тетраду порта RA. Младшую тетраду оставить без изменений.

## 4 Контрольные вопросы

- Из каких блоков состоит программа на ассемблере?
- В чём отличие директив и инструкций?
- Укажите синтаксис директив.
- Укажите синтаксис инструкций.
- Каково назначение литералов?

## Лабораторная работа 4

## Изучение системы программирования микроконтроллеров dsPIC33 с помощью языка программирования Си

**Цель работы:**

Изучить правила программирования микроконтроллера на языке программирования С, особенности использования языка при разработке программ. По заданному алгоритму составить программу, откомпилировать и исследовать ход её работы в симуляторе

**Порядок выполнения работы:**

- Изучить основные положения и конструкции языка Си.
- Разработать программу в соответствии с вариантом индивидуального задания.
- Отладить работу программы.
- Проследить ход выполнения программы в симуляторе.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

MPLAB C30 – это профессиональный компилятор языка Си высокого уровня. При установке компилятора происходит его полная интеграция со средой разработки MPLAB IDE.

Основные особенности компилятора C30:

- ANSI C совместимый, в комплекте поставляются стандартные библиотеки.
- поддержка типов 32-bit double и 64-bit double, 64-битного целого (long long).
- поддержка всех 16-битных семейств Microchip.

- поддержка смешанного кода Си + asm.
- поддержка расширенных настроек оптимизации кода.

- наличие встроенных (intrinsincs) функций для работы с DSP-ядром. Для каждого микроконтроллера всех поддерживаемых семейств компилятор

MPLAB C30 имеет в своём составе заголовочные файлы с описанием всех доступных регистров микроконтроллера с указанием их адресов. Символьные

имена регистров полностью совпадают с названиями, предложенными в документации на микроконтроллер, таким образом, обращение к регистрам микроконтроллера из кода программы происходит в простой, удобной и наглядной форме. Название заголовочного файла подразумевает модель микроконтроллера. Например, для микроконтроллера dsPIC33fj32mc204 подключение заголовочного файла с описанием всех его регистров на языке С будет выглядеть следующим образом:

```
#include <P33FJ32MC204.h>
```

Однако для реализации специфических функций микроконтроллера, компилятор поддерживает некоторые дополнительные возможности. Наиболее часто используемые из них:

определение атрибутов переменных;
определение атрибутов функций;

Inline-функции;
переменные в определённых регистрах;

целые размером в двойное слово.

**Определение атрибутов переменных**

Атрибуты переменных устанавливаются с помощью ключевого слова __attribute__

- задают такие специфические свойства переменной, как расположение по указанному адресу либо в указанной области памяти, выравнивание переменной, параметры загрузки начального значения, инициализации либо очистки значения переменной и прочее. Например, для объявления переменной целочисленного типа, расположенной по адресу 0x900 секции «mysection», необходимо воспользоваться следующей конструкцией:

```
int foo __attribute__((section("mysection"),address(0x900)));
```

**Определение атрибутов функций**

С помощью атрибутов функций объявляются сведения о функции, вызываемой основной программой, что помогает компилятору оптимизировать вызов функции и более тщательно проверять код. Атрибуты функции задаются так же с помощью ключевого слова __attribute__, за которым следует вспомогательная спецификация в двойных круглых скобках. С помощью атрибутов возможно задать абсолютный адрес функции,

поместить функцию в сегмент загрузки программной flash памяти, указать компилятору на функцию-обработчик прерывания и прочее. Например, объявление функции-обработчика прерывания таймера T1 с указанием компилятору не генерировать дополнительный код будет выглядеть следующим образом:

```
void __attribute__((interrupt, no_auto_psv))
_T1Interrupt( void ) { }
```

Inline-**функции**

Объявление функции с ключевым словом inline приводит к тому, что компилятор внедряет код этой функции в код вызывающих ее операторов. Это обычно ускоряет выполнение, устраняя накладные расходы на вызов. Кроме того, если любые фактические параметры — константы, знание их величин позволяет упрощать код на этапе компиляции, чтобы сократить объем включаемого кода. Объявление inline-функции происходит следующим образом:

```
inline int inc (int *a)
{
        (*a)++;
}
```

**Переменные в определённых регистрах**

Компилятор позволяет помещать несколько глобальных переменных в определенные аппаратные регистры. Глобальные регистровые переменные резервируют регистры на все время исполнения программы. Для определения глобальной регистровой переменной необходимо воспользоваться следующей конструкцией:

```
register int *foo asm ("w8");
```

Здесь w8 - имя регистра, который будет использован.

**Целые размером в двойное слово**

Компилятор поддерживает типы данных для целых, которые в два раза длиннее чем long int. Тип таких переменных long long int для целого со знаком или unsigned long long int для целого без знака. Для записи констант типа long long int, добавляется суффикс LL к целому. Для записи констант типа unsigned long long int, добавляется суффикс ULL к целому.

Данные типы можно использовать в арифметике подобно любым другим

целым типам. Сложение, вычитание и поразрядные логические операции для этих типов реализованы открытым кодом, но деление и сдвиги реализованы не на основе открытого кода. Операции, не использующие для реализации открытый код, требуют специальных библиотечных подпрограмм, которые поставляются с компилятором.

Так же компилятор MPLAB C30 позволяет устанавливать биты конфигурации микроконтроллера в коде программы следующими директивами:

```
_FOSCSEL(st
ate);
_FOSC(state);
_FWDT(state)
; _FICD(state);
```

Синтаксис использования директив аналогичен синтаксису вызова функций, однако установка битов конфигурации должна осуществляться вне кода какой-либо из функций.

**Главный цикл**

Язык С предполагает, что по завершении функции main() управление должно возвращаться операционной системе. Однако во встраиваемых приложениях требуется,

чтобы программа выполнялась бесконечно с момента включения питания и до его выключения, следовательно, приложения разрабатываются с тем расчѐтом, что главный цикл программы будет выполняться бесконечно.

Для реализации данного принципа с помощью языка С наибольшее распространение получил способ использования цикла while:

```
while (1)

{

        // код программы

}
```

Всё, что помещено в теле цикла, будет повторяться до тех пор, пока условие цикла возвращает значение «истина». А так как условие цикла представляет собой константу, возвращающую «истина», тело цикла будет выполняться бесконечно.

## 2 Пример выполнения работы

*Задача:* Разработать программу для разбиения числа на цифры. Проверить корректность выполнения алгоритма в симуляторе. Промежуточные значения переменных на каждом шаге занести в таблицу.

*Анализ задачи:* Операция разбиения числа на цифры оформлена в виде функции. Передача значения числа в функцию осуществляется через параметр типа unsigned int. Результат разбиения возвращается через указатель на массив типа char, каждый элемент которого содержит цифру числа.

Листинг программы:

```
char* Convert(unsigned int a)
{
	char digits[5] = {0, 0, 0, 0, 0};
	int i = 0;
	while (a)
	{
		digits[i] = a %
		10; i = i + 1;
		a = a / 10;
	}
	return &(digits[0]);
}
void main()
{
	char digits = Convert(53417);
}
```

Таблица 2.1

Значение переменных во время выполнения программы

| *Шаг* | *Значение переменой* | | |
|---|---|---|---|
| | *a* | *i* | *digits[5]* |
| 1 | 0xD0A9 (53417) | 0 | {0, 0, 0, 0, 0} |
| 2 | 0x14DD (5341) | 1 | {0, 0, 0, 0, 7} |
| 3 | 0x0216 (534) | 2 | {0, 0, 0, 1, 7} |
| 4 | 0x0035 (53) | 3 | {0, 0, 4, 1, 7} |
| 5 | 0x0004 (5) | 4 | {0, 3, 4, 1, 7} |
| 6 | 0x0000 (0) | 5 | {5, 3, 4, 1, 7} |

## 3 Варианты индивидуальных заданий к лабораторной работе

Разработать программу в соответствии с вариантом задания. Проверить корректность выполнения алгоритма в симуляторе. Промежуточные значения переменных на каждом шаге занести в таблицу.

3.1 Вычислить значение функции $Y(x) = \begin{cases} x - 25, x > 128 \\ x^2, x \leq 128 \end{cases}$.

3.2 Вычислить значение функции $Y(x) = \begin{cases} x^2, x - \text{четное} \\ \sqrt{x}, x - \text{нечетное} \end{cases}$.

3.3 Вычислить значение функции $Z(x, y) = \begin{cases} x + y, x > y \\ x * y, x \leq y \end{cases}$.

3.4 Найти наибольшее из трёх чисел X, Y, Z.

3.5 Найти наименьшее из трёх чисел X, Y, Z.

3.6 Найти среднее арифметическое из трёх чисел X, Y, Z.

3.7 Вычислить количество чисел в массиве $\{X_i\}$, больших Z.

3.8 Вычислить сумму чисел в массиве $\{X_i\}$, меньших Z.

## 4 Контрольные вопросы

1. Укажите особенности компилятора MPLAB C30.
2. Для чего применяются атрибуты переменных?
3. Для чего применяются атрибуты функций?
4. Каким образом объявляется функция обработки прерывания?
5. Каким образом можно разместить переменную по абсолютному адресу?
6. Каким образом устанавливаются биты конфигурации?

# Лабораторная работа 5

## Изучение устройств ввода-вывода дискретных сигналов в микропроцессорных системах управления

**Цель работы:**

Изучить структуру и особенности работы портов микроконтроллера, схему подключения входных и выходных дискретных сигналов к микроконтроллеру, особенности программирования ввода-вывода дискретных сигналов на языке программирования. Составить программу ввода, обработки по заданному алгоритму и вывода дискретных сигналов, записать в память программ микроконтроллера и выполнить.

**Порядок выполнения работы:**

- Изучить теоретические вопросы, связанные с функционированием дискретных входов-выходов.
- Изучить принципиальную электрическую схему к лабораторной работе.
- Разработать программу в соответствии с индивидуальным заданием.
- Отладить программу в среде MPLAB IDE.
- Загрузить программу в учебный стенд.
- Исследовать работу дискретных входов и выходов.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

#### 1.1 Параллельные порты ввода-вывода микроконтроллера dsPIC33fj32mc204

Все выводы микроконтроллера (кроме выводов питания Vdd, Vss, AVdd, AVss, VCap и вывода сброса MCLR) могут использоваться как периферийными модулями, так и параллельными портами ввода/вывода. Все вводные линии портов имеют триггер Шмидта по входу для исключения влияния электромагнитных помех и шумов.

Линии ввода/вывода микроконтроллера разделены на три порта: RA, RB, RC. Подавляющее большинство линий ввода/вывода всех портов имеют дополнительные функции и могут использоваться различными периферийными модулями микроконтроллера. Блок-схема структуры линии порта в таком случае приведена на рис. 1.1.

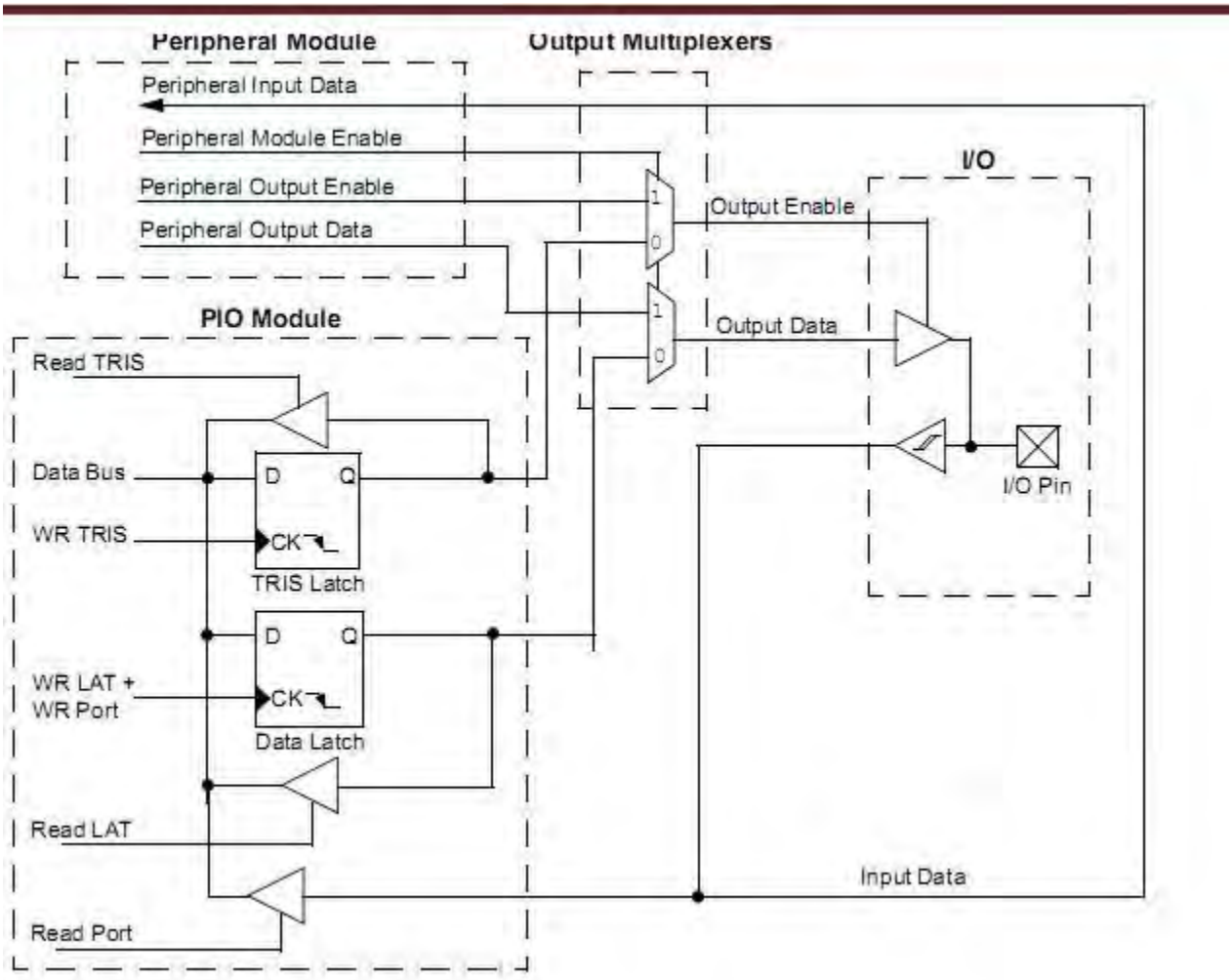



Рисунок 1.1 – Блок-схема структуры линии порта, объединённой с выводами периферийных функций

Мультиплексор выбирает, каким образом функционирует линия порта (в качестве части периферийного модуля, либо в качестве линии параллельного порта ввода/вывода). Таким образом, перед использованием таких линии в качестве линий порта ввода/вывода необходимо предварительно отключить соответствующие периферийные устройства. Например, при использовании порта RA2 как линии ввода, необходимо отключить дополнительную функцию порта RA2 (выход тактирующего сигнала внутреннего генератора) следующей директивой:

```
_FOSC(OSCIOFNC_ON & POSCMD_NONE)
```

Для работы с портами каждый из них имеет 3 специальных регистра:

TRISx[1] – регистр направления данных – задаёт каким образом используется линия порта (как вход либо как выход). При установке соответствующего бита в 1 линия порта будет сконфигурирована как вход.

[1] Действительные названия регистров получаются подстановкой названия порта вместо символа x. Соответственно, регистры порта A называются TRISA, LATA, PORTA, порта B - TRISB, LATB, PORTB, порта C - TRISC, LATC, PORTC.

LATx – регистр выводов порта – при установке соответствующего бита в 1 данного регистра позволяет установить высокий уровень сигнала на выходе линии порта, при установке соответствующего бита в 0 – низкий уровень сигнала.

PORTx – регистр состояния порта – чтение соответствующего бита из данного регистра позволяет получить состояние сигнала (высокий либо низкий уровень) на входе линии порта.

*Примечание.* Для того, чтобы установка бита регистра LATx приводила к соответствующему изменению состояния линии вывода порта, необходимо чтобы эта линия была предварительно сконфигурирована как выход установкой требуемого бита в регистре TRISx. Аналогично, чтобы чтение бита регистра PORTx отображало действительное состояние линии ввода порта, необходимо чтобы эта линия была предварительно сконфигурирована как вход установкой требуемого бита в регистре TRISx.

Таким образом, настройка линии 2 порта RA на вход осуществляется следующим образом:

```
TRISA |= (1 << 2);
```

А настройка линии 15 порта RB на выход осуществляется следующим образом:

```
TRISB &= ~(1 << 15);
```

Так же MPLAB позволяет обращаться напрямую к битам регистров портов. Используя такой приём, вышеприведённые примеры будут выглядеть следующим образом:

```
TRISAbits.TRISA2 = 1;
```

```
TRISBbits.TRISB15 = 0;
```

Кроме того, некоторые линии портов микроконтроллера могут быть сконфигурированы как выход с открытым коллектором. В таком случае линии порта функционирует как транзистор, эмиттер которого подключён к земле, а

коллектор к выводу линии порта. Подключение внешнего подтягивающего резистора к такому выводу позволяет получить на выходе линии напряжение, отличное от напряжения питания микроконтроллера, что в свою очередь необходимо для согласования микросхем с различными напряжениями питания. Настроить как выходы с открытым коллектором можно те выводы микроконтроллера, которые обозначены как выводы, допускающие подачу на них 5-вольтового напряжения. Настройка вывода микроконтроллера как выход с открытым коллектором осуществляется установкой соответствующего бита в регистрах
ODCx[2]. Например, настройка линии 8 порта RA как выхода с открытым коллектором будет осуществляться следующим образом:

```
ODCAbits.ODCA8 = 1;
```

После настройки линии порта как выхода с открытым коллектором управление состоянием транзистора рекомендуется осуществлять с помощью регистра TRISx. Для открытия транзистора следует установить соответствующий бит регистра, для закрытия транзистора – сбросить. Таким образом, управление транзистором линии 8 порта RA (настроенной как выход с открытым коллектором), будет осуществляться следующем образом:

```
TRISAbits.TRISA8 = 1; // Открытие транзистора

TRISAbits.TRISA8 = 0; // Закрытие транзистора
```

## 2 Электрическая принципиальная схема к лабораторной работе

Для ввода дискретной информации в микроконтроллер широко применяются различные переключатели, кнопки и клавиатуры, либо иные дискретные датчики. Дискретными выходами микроконтроллер управляет различными исполнительными устройствами, работающими по принципу включено/выключено.

На рис. 2.1 приведена электрическая принципиальная схема к лабораторной работе.

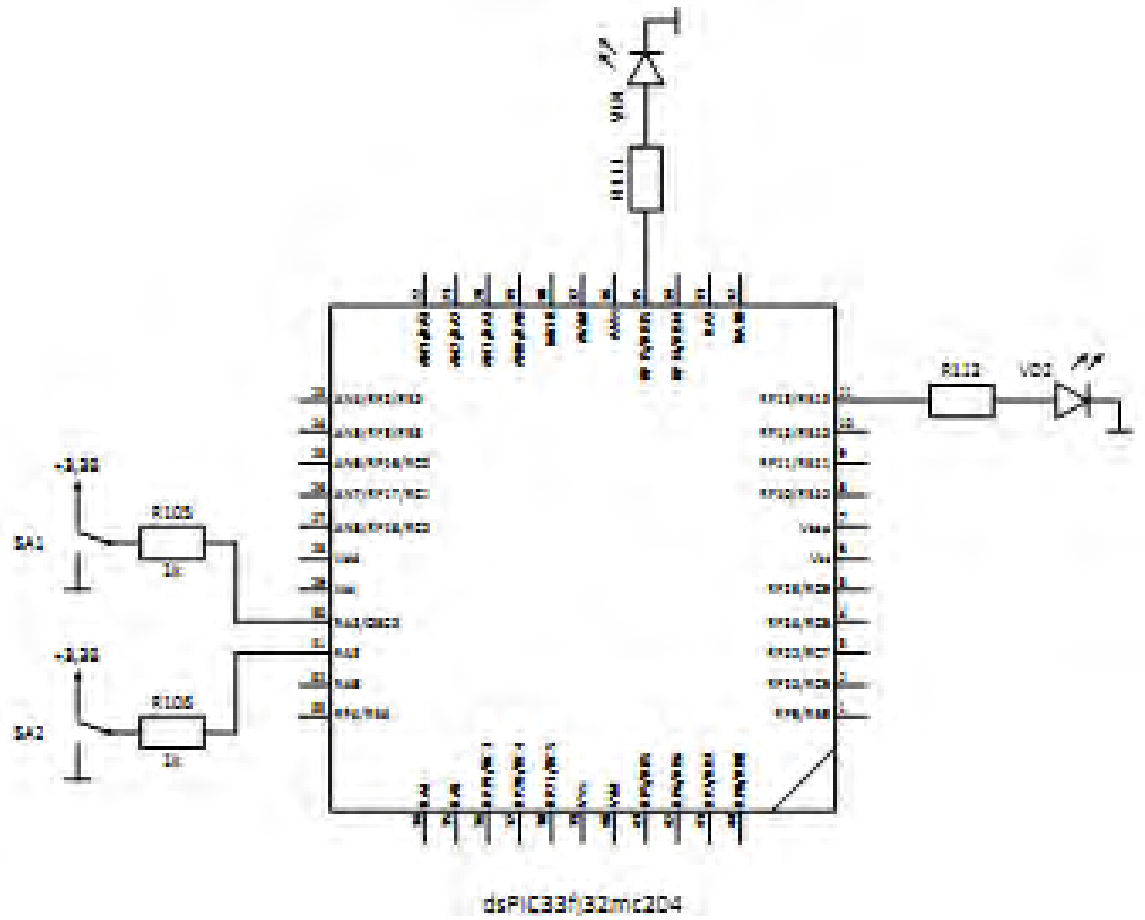



Рисунок 2.1 – Электрическая принципиальная схема к лабораторной работе

В схеме два дискретных датчика оформлены в виде двух переключателей SA1 и SA2, подключенных к выводам RA2 и RA3 микроконтроллера. Два дискретных выхода оформлены в виде двух светодиодов VD1 и VD2, подключенных к выводам RB15 и RB13 микроконтроллера соответственно.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую отображать на VD1 состояние SA1 и на VD2 состояние SA2.

### Ввод дискретных сигналов

Прежде чем приступить к обработке входного дискретного сигнала, необходимо инициализировать используемый порт – настроить линию порта как вход. Так же в случаях, если требуемая линия порта может использоваться для каких либо других периферийных функций, необходимо их отключить.

Непосредственно обработка сигнала от дискретного датчика подразумевает либо определение уровня сигнала в текущий момент времени,

либо ожидание появления сигнала требуемого уровня. Конкретная программная реализация процедуры зависит от того, каким образом датчик подключен к микроконтроллеру.

Например, при подключении датчика к линии бита 2 порта RA программа определения уровня сигнала в текущий момент времени будет иметь вид:

```
#include <P33FJ32MC204.h>
_FOSC(OSCIOFNC_ON & POSCMD_NONE)   // отключение дополнительной
                                   // функции порта RA2 – выход
                                   // тактирующего сигнала
                                   // внутреннего генератора
void main()
{
    TRISAbits.TRISA2 = 1; // настройка порта RA2 на вход
    if (PORTAbits.RA2)
    {
        • часть программы, выполняемой при ВЫСОКОМ уровне
        • сигнала на входе контроллера
    }
    if (!PORTAbits.RA2)
    {
        3.1  часть программы, выполняемой при НИЗКОМ уровне
        3.2  сигнала на входе контроллера
    }
}
```

Программа ожидания требуемого уровня сигнала на входе микроконтроллера при данном подключении будет иметь вид:

```
#include <P33FJ32MC204.h>
_FOSC(OSCIOFNC_ON & POSCMD_NONE)   // отключение дополнительной
                                   // функции порта RA2 – выход
                                   // тактирующего сигнала
                                   // внутреннего генератора
void main()
{
    TRISAbits.TRISA2 = 1; // настройка порта RA2 на вход
    while (PORTAbits.RA2);       // ожидание НИЗКОГО уровня сигнала
    …
    while (!PORTAbits.RA2);      // ожидание ВЫСОКОГО уровня сигнала
    …
}
```

**Вывод дискретных сигналов**

Аналогично, прежде чем приступить к управлению выходным дискретным сигналом, необходимо инициализировать используемый порт – настроить линию порта как выход. Так же в случаях, если требуемая линия порта может использоваться для каких либо других периферийных функций, необходимо их отключить.

Для управления дискретным выводом микроконтроллера на соответствующей выходной линии порта необходимо сформировать логический сигнал 0 или 1, что реализуется командами вывода непосредственного операнда, содержащего в требуемом бите значение 0 или 1. Таким образом, при подключении исполнительного устройства к линии бита 15 порта RB программа работы с дискретным выходом микроконтроллера будет иметь вид:

```
#include <P33FJ32MC204.h>
void main()
{
        TRISBbits.TRISB15 = 0;        // настройка порта RB15 на выход
        LATBbits.LATB15 = 1; // установка ВЫСОКОГО уровня сигнала
        …
        LATBbits.LATB15 = 0; // установка НИЗКОГО уровня сигнала
        …
}
```

Блок-схема алгоритма решения задачи представлена на рис. 3.1.

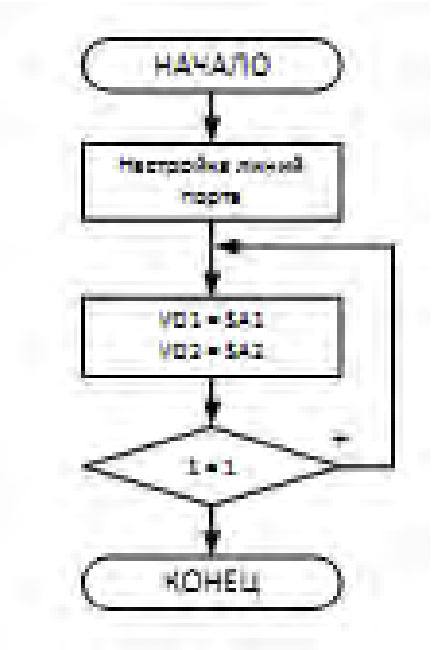



Рисунок 3.1 – Блок-схема алгоритма решения задачи

Листинг программы для решения задачи:

```
#include <P33FJ32MC204.h>
_FOSC(OSCIOFNC_ON & POSCMD_NONE)   // отключение дополнительной
                                   // функции порта RA2 – выход
                                   // тактирующего сигнала
                                   // внутреннего генератора
void main()
{
    TRISBbits.TRISB15 = 0;      // настройка порта RB15 на выход
    TRISBbits.TRISB13 = 0;      // настройка порта RB13 на выход
    TRISAbits.TRISA2 = 1;       // настройка порта RA2 на вход
    TRISAbits.TRISA3 = 1;       // настройка порта RA3 на вход
    while (1)
    {
        LATBbits.LATB15 = PORTAbits.RA2;
        LATBbits.LATB13 = PORTAbits.RA3;
    }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, выполняющую следующие действия:

1. Если SA1 = 0, то VD1 = 0 и VD2 = 0; иначе, если SA2 = 0, то VD1=1, VD2 = 0, если SA2 = 1, то VD1=0, VD2 = 1.
2. Если SA1 = 1 и SA2 = 0 то VD1 = 0 и VD2 = 0, если SA1 = 0 и SA2 = 1, то VD1=1, VD2 = 1, если SA1 = SA2, то VD1=0, VD2 = 1.
3. Если SA1 = 1, то VD1 = 0 и VD2 = 0; иначе, если SA2 = 1, то VD1 = 1, VD2 = 0, если SA2 =0, то VD1=0, VD2 = 1.
4. Если SA1 = 1 и SA2 = 1 то VD1 = 0 и VD2 = 0, если SA1 = 0 и SA2 = 0, то VD1 = 1, VD2 = 1, если SA1 ≠ SA2, то VD1 = 1, VD2 = 0.

## 5 Контрольные вопросы

1. Дайте определение дискретного сигнала.
2. Приведите пример устройства либо механизма, выходной сигнал которого является дискретным.
3. Приведите пример устройства либо механизма, управление которым осуществляется дискретным сигналом.
4. Каким образом настраивается линия порта микроконтроллера на вход либо на выход?
5. Как организуется ввод дискретного сигнала?
6. Как организуется вывод дискретного сигнала?

# Лабораторная работа 6

## Реализация дополнительных портов ввода-вывода дискретных сигналов в микропроцессорных системах управления

**Цель работы:**

Изучить структуру и особенности работы различных схемы внешних портов ввода-вывода дискретных сигналов. Составить программу ввода, обработки по заданному алгоритму и вывода дискретных сигналов, записать в память программ микроконтроллера и выполнить.

**Порядок выполнения работы:**

- Изучить теоретические вопросы, связанные с функционированием дополнительных дискретных входов-выходов.
- Изучить принципиальную электрическую схему к лабораторной работе.
- Разработать программу в соответствии с индивидуальным заданием.
- Отладить программу в среде MPLAB IDE.
- Загрузить программу в учебный стенд.
- Исследовать работу дополнительных дискретных входов и выходов.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Способы построения внешних портов ввода-вывода

Микроконтроллеры семейства dsPIC33 имеют в своем составе несколько параллельных портов ввода вывода. Однако существуют приложения, в которых требуется большее число портов ввода/вывода.

Существуют различные схемы расширения портов ввода-вывода. Часть из них основана на использовании сдвиговых регистров. Сдвиговый регистр представляет собой набор ячеек, которые последовательно связаны между собой в одном направлении. Таким образом, сдвиговый (или последовательный) регистр служит для преобразования последовательного кода в параллельный либо наоборот. Применение последовательного кода связано с необходимостью передачи большого количества двоичной информации по ограниченному количеству соединительных линий.

Для управления типовыми сдвиговыми регистрами достаточно трёх выводов: тактирование (SCK), линия данных (SER) и вход защёлки (LE). Рассмотрим принцип работы сдвиговых регистров на примере преобразования последовательного кода в параллельный в сдвиговом регистре вывода HC595 (рис. 1.1).

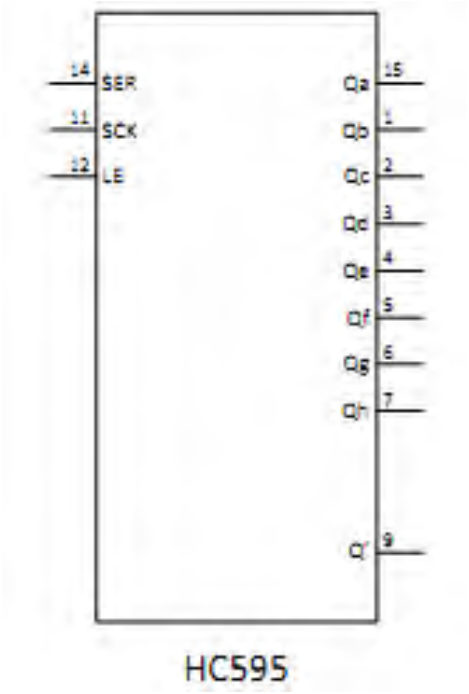



Рис. 1.1. Микросхема сдвигового регистра вывода HC595

Отдельные биты двоичной информации последовательно подаются на вход сдвигового регистра SER. Каждый бит сопровождается отдельным тактовым импульсом синхронизации, который поступает на вход синхронизации сдвигового регистра SCK. После поступления первого тактового импульса логический уровень, присутствующий на входе SER, запоминается в первой ячейке регистра.

После поступления второго тактового импульса логический уровень, присутствующий в первой ячейке (установленный в предыдущем шаге), передаётся во вторую ячейку регистра. Одновременно следующий бит входного последовательного кода запоминается в первом триггере сдвигового регистра.

Описанная процедура повторятся количество раз, соответствующее количеству выводов сдвигового регистра.

После того, как весь код записан во внутренние ячейки сдвигового регистра, по тактовому импульсу на входе защёлки LE происходит выставление значений логических уровней сигналов на параллельных выходах Qa…Qh сдвигового регистра в соответствии со значениями внутренних ячеек.

Временная диаграмма работы последовательного регистра вывода представлена на рис. 1.2.

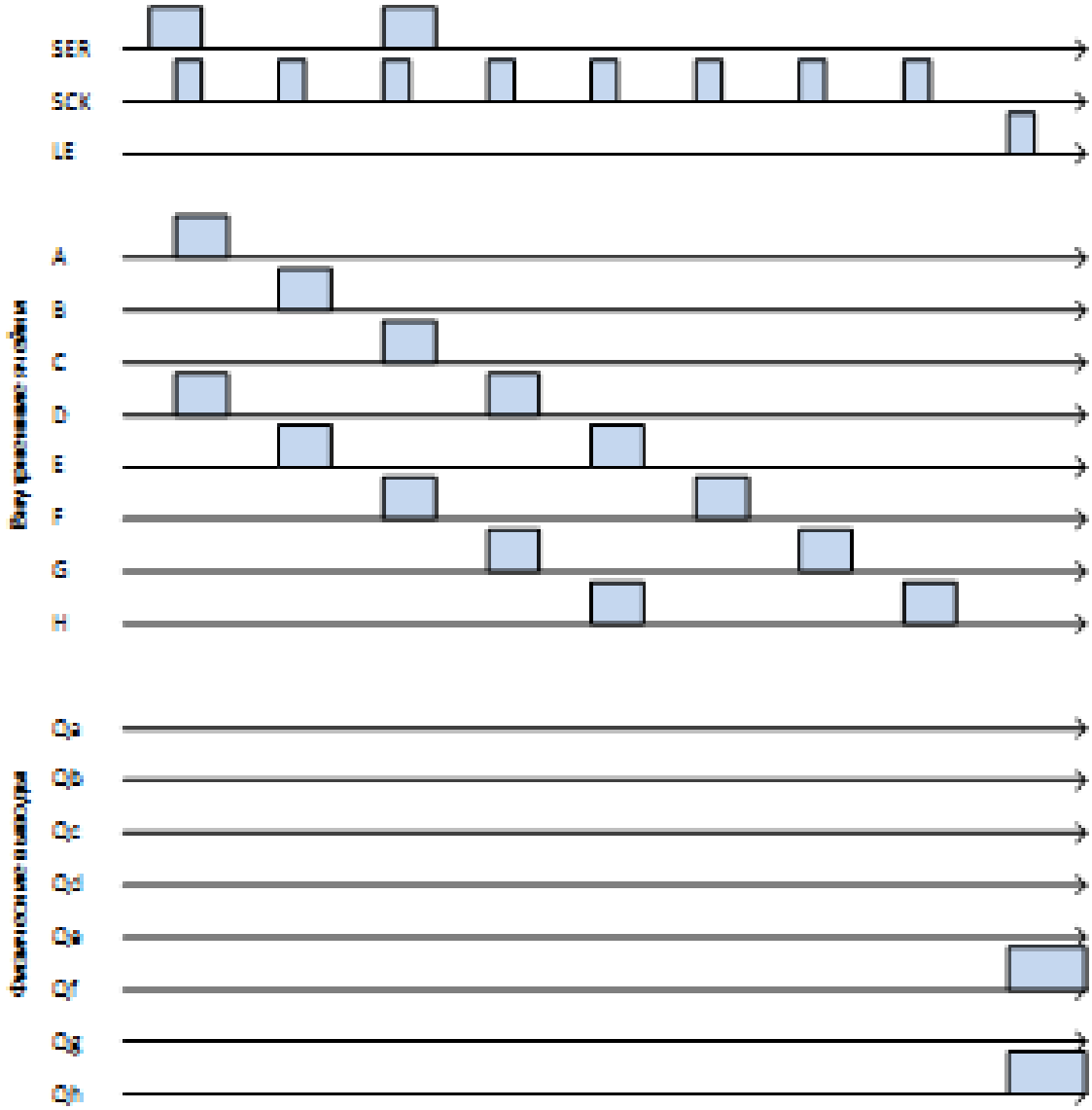



Рис. 1.2. Временная диаграмма работы последовательного регистра вывода

Так же к достоинствам сдвиговых регистров следует отнести возможность каскадного соединения большого числа регистров друг с другом. При этом на последовательный вход последующего сдвигового регистра подаётся выход Q’ предыдущего, а входы тактирования и данных всех регистров объединяются. Таким образом, такое соединение регистров не приводит к увеличению числа управляющих линий.

Сдвиговый регистр ввода HC165 (рис. 1.3) функционирует аналогично. Временная диаграмма его работы представлена на рис. 1.4.

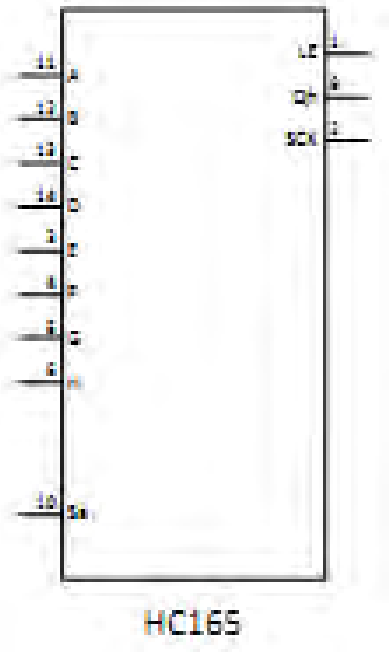



Рис. 1.3. Микросхема сдвигового регистра вывода HC595

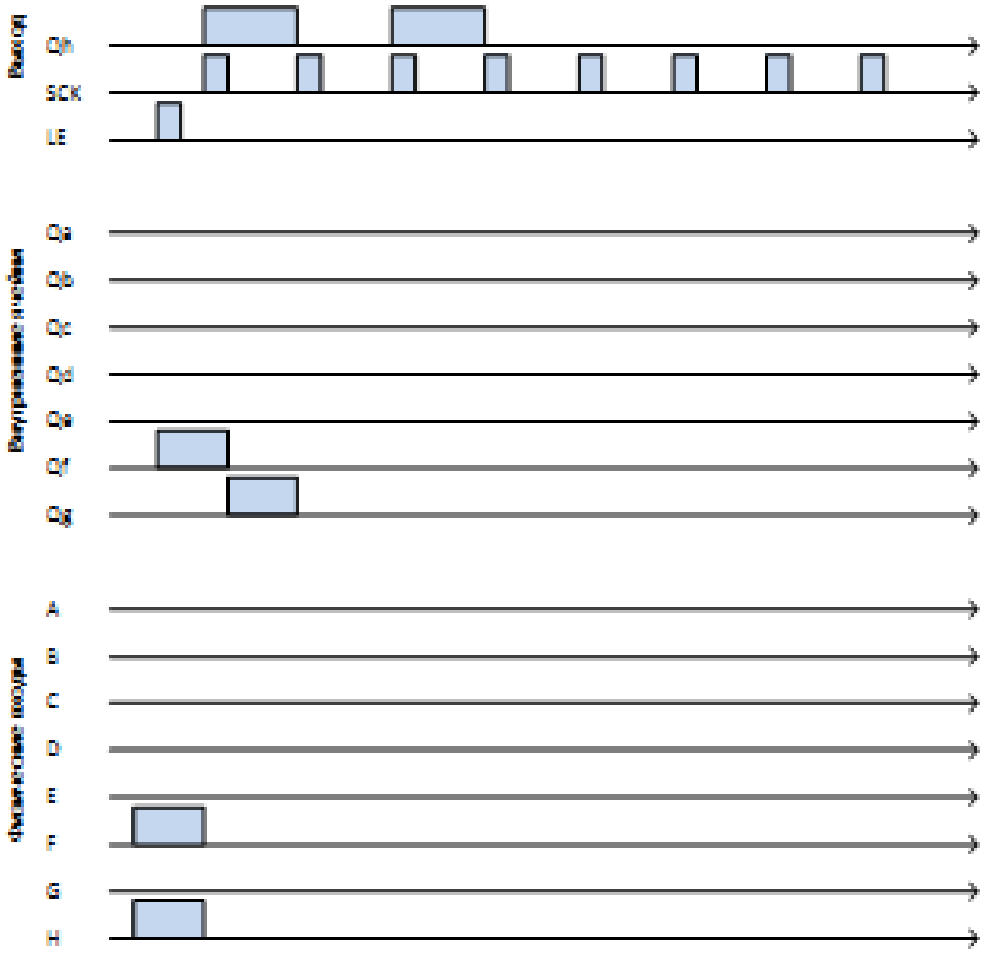



Рис. 1.4. Временная диаграмма работы последовательного регистра ввода

## 2 Электрическая принципиальная схема к лабораторной работе

В схеме использован сдвиговый регистр вывода HC595, к параллельным выходам которого подключены светодиоды VD3..VD10, и сдвиговый регистр ввода HC165, к параллельным входам которого подключены тумблеры SA3..SA10. Управляющие сигналы LE и SCK объединены и подключены к выводам RB5 и RC4 микроконтроллера соответственно. Вход данных регистра вывода HC595 подключен к выводу RC5, выход данных регистра ввода HC165 подключен к выводу RA4.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую отображать на VD3..VD10 состояние SA3..SA10.

Блок-схема алгоритма решения задачи представлена на рис. 3.1.

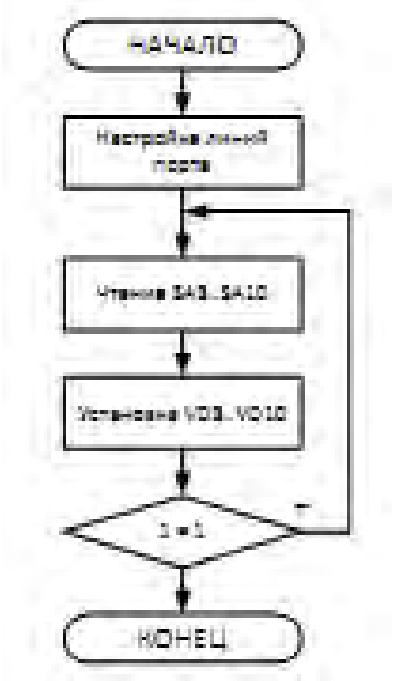



Рисунок 3.1 – Блок-схема алгоритма решения задачи  Листинг программы для решения задачи:

```
#include <P33FJ32MC204.h>

void Serial_Init()                    // настройка линий порта
{
    TRISCbits.TRISC4 = 0;             // Выход SCK (RC4)
    TRISBbits.TRISB5 = 0;             // Выход LE (RB5)
```

```
        TRISCbits.TRISC5 =                              (RC5 -
        0;                          // Выход    SER    )    VD
        TRISAbits.TRISA4 =                              (RA4 -
        1;                          // Вход     SDI    )    SA
}

                                    // Отправление данных в регистр
void Serial_Send(char VDs)          HC595
{
        char c;
        for (c = 0; c < 8; c++)
        {
                установка требуемого логического уровня

                на последовательном входе SER регистра HC595

                начиная с последнего бита
                if ((VDs & (1 << (7 - c))) != 0)

                {

                        LATCbits.LATC5 = 1;
                }
                else
                {
                        LATCbits.LATC5 = 0;
                }
                // формирование синхроимпульса на
                входе SCK LATCbits.LATC4 = 1;
                LATCbits.LATC4 = 0;
        }
        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
        LATBbits.LATB5 = 0;
}

char Serial_Read()                  // Чтение данных из регистра HC165
{
        • формирование синхроимпульса
        на LE LATBbits.LATB5 = 0;
        LATBbits.LATB5 = 1;
        • чтение логического уровня
        • на последовательном выходе Qh регистра
        HC165 char SAs = 0x00;
        char c;
```

```
        for (c = 0; c < 8; c++)
        {
                o чтение логического уровня
                p на последовательном выходе Qh регистра HC165
                q сдвигая предыдущие данные влево
                SAs <<= 1;
                if (PORTAbits.RA4)
                {
                        SAs |= 0x01;
                }
                // формирование синхроимпульса на
                входе SCK LATCbits.LATC4 = 1;
                LATCbits.LATC4 = 0;
        }
        return SAs;
}
void main()
{
        Serial_Init();          // настройка линий порта
        while (1)
        {
                char state =
                Serial_Read();
                Serial_Send(state);
        }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнять определенные действия с дискретными светодиодными индикаторами VD3..VD10 в зависимости от состояния дискретных датчиков SA3..SA10:

- Если SA = 01010101, то на VD вывести «0»; если SA = 00011000, то на VD вывести «5» (в двоичном коде).
- Если SA = 01010101, то на VD вывести «8»; если SA = 10000001, то на VD вывести «2» (в двоичном коде).
- Отобразить на VD количество включенных SA, добавляя зажженный светодиод справа налево, если включается дополнительный SA.
- Отобразить на VD количество включенных SA, добавляя зажженный светодиод слева направо, если включается дополнительный SA.
- Отобразить на VD количество выключенных SA, добавляя зажженный светодиод слева направо, если включается дополнительный SA.
- Отобразить на VD количество выключенных SA, добавляя зажженный

светодиод справа налево, если включается дополнительный SA.

- При любом изменении состояния SA перемещать зажженный светодиод на VD справа налево.
- При любом изменении состояния SA перемещать зажженный светодиод на VD слева направо.

## 5 Контрольные вопросы

1. Дайте определение дискретного сигнала.
2. Приведите пример устройства либо механизма, выходной сигнал которого является дискретным.
3. Расскажите способы увеличения дискретных портов ввода вывода микроконтроллера.
4. Расскажите принцип работы сдвигового регистра.
5. Как организуется ввод дискретного сигнала при использовании сдвигового регистра?
6. Как организуется вывод дискретного сигнала при использовании сдвигового регистра?

# Лабораторная работа 7

## Изучение системы прерываний микроконтроллера

**Цель работы:**

Изучить механизм прерываний микроконтроллера, особенности настройки и программирования функций обработки прерываний микроконтроллера на языке программирования Си. Составить программу ввода, обработки по заданному алгоритму и вывода дискретных сигналов, записать в память программ микроконтроллера и выполнить.

**Порядок выполнения работы:**

1. Изучить теоретические вопросы, связанные с механизмом прерываний микроконтроллера.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу прерываний.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Система прерываний микроконтроллера dsPIC33fj32mc204

Прерывание – сигнал, сообщающий о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается, и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

Микроконтроллеры семейства dsPIC33 имеют развитую систему прерываний, позволяющую использовать это устройство в приложениях реального времени различной степени сложности. Модуль прерываний микроконтроллеров dsPIC33 обладает следующими характеристиками:

1. обеспечивает обработку 8 немаскируемых системных прерываний;
2. позволяет задавать до семи уровней приоритетов пользовательских прерываний;
3. позволяет задавать до 126 векторов прерываний в таблице векторов прерываний IVT. При этом каждому прерыванию назначается уникальный вектор;
4. позволяет настроить дополнительную таблицу векторов прерываний (AIVT), которую можно использовать при отладке программ;

5. фиксированное время запуска и выхода из процедуры обработки прерывания. Наиболее часто используемыми событиями, вызывающими прерывания, являются:
6. совпадение либо переполнение значения счётчика таймера;
7. появление сигнала требуемого уровня либо изменение величины сигнала на входе линии порта;
8. приём данных, отправка данных модулей приёмо-передатчиков;
9. завершение преобразования модуля АЦП.

Каждый источник прерывания имеет собственный вектор в таблице, расположенной в программной памяти. Внутри таблицы прерывания имеют естественный приоритет: при одновременном возникновении двух прерываний приоритет имеет то, чей вектор имеет меньший адрес. Вектора представляют собой 24-битное слово программы, в котором должна быть расположена команда перехода на сервис обработчика прерывания.

Пользователь может изменить естественный приоритет, назначив источнику прерывания искусственный приоритет. При возникновении прерывания в программный счетчик записывается адрес вектора прерывания, а приоритет CPU-ядра становится равным приоритету возникшего прерывания. Это позволяет организовывать гибкую систему вложенных прерываний.

Возможно назначение приоритета каждому прерыванию, количество возможных уровней приоритета равно 8. CPU ядро может иметь приоритет от 0 до 15, уровни 8 – 15 зарезервированы для аппаратных исключений.

Аппаратные исключения – это определенный вид немаскируемых прерываний, которые генерирует CPU-ядро. Условно исключения можно разделить на программные и аппаратные.

К программным относятся исключения, которые позволяют продолжить выполнение работы после программного сброса флага исключения. Программные исключения имеют приоритет от 8 до 12. К программным исключениям относятся: исключение АЛУ (которое генерируется ядром, например, при делении на 0) и ошибка стека.

При возникновении аппаратного исключения программа не может продолжить работу до тех пор, пока ошибка, которая вызвала исключение, не будет устранена – флаг аппаратного исключения не может быть сброшен программно. Аппаратные исключения имеют приоритет от 13 до 15. К аппаратным исключения относятся: ошибка адреса (не выровненный доступ к памяти программ или памяти данных) и ошибка тактового генератора (нестабильная работа PLL, отсутствие внешней тактовой частоты).

Система прерываний имеет фиксированное время реакции – 5 командных тактов. Возврат из прерывания осуществляется за 3 командных такта. Следует заметить, что при входе в прерывание часть статус-регистра SR автоматически сохраняется в стеке, это позволяет уменьшить объем кода и время выполнения обработчика прерывания. Все флаги прерываний устанавливаются вне зависимости от того, разрешено прерывание или нет.

Таблица векторов прерываний IVT размещается в памяти программ, начиная с адреса 0x000004, и содержит 126 векторов прерываний, из которых 8 используются системными прерываниями, а остальные 118 могут использоваться периферийными модулями. Системные прерывания позволяют выполнить обработку серьезных аппаратно программных сбоев в системе, поэтому их нельзя запретить или замаскировать.

Список векторов наиболее часто используемых пользовательских прерываний представлен в таблице 1.1. Каждый вектор прерывания содержит 24-битный адрес, по которому располагается соответствующая функция-обработчик прерывания (Interrupt Service Routine, ISR).

Таблица 1.1

Пользовательские прерывания

| *Номер вектора* | *Адрес вектора* | *Источник прерывания* |
|---|---|---|
| 8 | 0x000014 | INT0 – Внешнее прерывание 0 |
| 9 | 0x000016 | IC1 – Модуль захвата таймера T1 |
| 10 | 0x000018 | OC1 – Совпадение таймера T1 |
| 11 | 0x00001A | T1 – Таймер T1 |
| 13 | 0x00001E | IC2 – Модуль захвата таймера T2 |
| 14 | 0x000020 | OC2 – Совпадение таймера T2 |
| 15 | 0x000022 | T2 – Таймер T2 |
| 16 | 0x000024 | T3 – Таймер T3 |
| 19 | 0x00002A | U1RX – Приёмник модуля UART1 |
| 20 | 0x00002C | U1TX – Передатчик модуля UART1 |
| 21 | 0x00002E | ADC1 – Модуль АЦП1 |
| 28 | 0x00003C | INT1 – Внешнее прерывание 1 |
| 37 | 0x00004E | INT2 – Внешнее прерывание 2 |

Любое пользовательское прерывание может быть разрешено или запрещено посредством установки или сброса соответствующего бита в одном из регистров разрешения прерываний IECx. Установка бита соответствующего прерывания в 1 разрешает прерывание, а сброс этого бита — запрещает. В микроконтроллерах dsPIC33 предусмотрен ряд механизмов,

позволяющих разрешить или запретить все прерывания одновременно (кроме, естественно, немаскируемых прерываний).

При сбросе микроконтроллера все биты разрешения прерываний сбрасываются в 0, запрещая генерацию прерываний, поэтому программа пользователя должна сама устанавливать необходимые разрешения.

Каждому пользовательскому прерыванию может быть присвоен тот или иной уровень приоритета. Биты управления приоритетом располагаются на трех младших позициях каждой тетрады регистров IPCn (четвёртые биты тетрад не используются и читаются как 0). Соответственно уровень приоритета каждого прерывания может изменяться от 1 (самый низкий приоритет) до 7 (наивысший приоритет). Если все биты приоритета прерывания сброшены в 0, то прерывание запрещено. По умолчанию (после сброса микроконтроллера) приоритеты пользовательских прерываний устанавливаются равными 4.

Контроллер прерываний микроконтроллеров семейства dsPIC имеет 22 регистра:

- INTCON1 – предназначен для настройки и управления системными немаскируемыми прерываниями.
- INTCON2 – предназначен для настройки событий возникновения внешних прерываний.
- IFSx[3] – регистр флагов прерываний. Каждому источнику прерываний соответствует определённый бит данного регистра, который устанавливается аппаратнопериферией при возникновении условия прерывания и сбрасывается программно функцией-обработчиком прерывания.
- IECx – регистр, содержащий биты разрешений прерываний. Данный регистр используется для индивидуальной настройки разрешения прерывания.
- IPCx – регистр приоритета прерываний.
- INTTREG – регистр настройки приоритета системных прерываний.

При возникновении разрешённого прерывания происходит сохранение текущего значения счётчика команд в стеке, и далее управление передаётся процедуре обработки прерывания. В этой процедуре флаг возникшего прерывания должен быть очищен программно, в противном случае произойдёт повторный вызов процедуры. После завершения обработки прерывания выход из процедуры должен осуществляться с помощью специальной инструкции RETFIE. При этом происходит автоматическое восстановление значения счётчика команд, значения регистра SRL и установка прежнего приоритета ядра.

### 1.2 Внешние прерывания

Микроконтроллер dsPIC33fj32mc204 может генерировать прерывания при изменении внешнего сигнала на следующих входных линиях:

1. вход INT0;
2. входы CN0..CN30.

Вход INT0 имеет схему определения фронта сигнала. Регистр INTCON2 имеет бит INT0EP, с помощью которого можно выбирать, по какому из фронтов (положительному либо отрицательному) сигнала на линии INT0 будет генерироваться прерывание

(_INT0Interrupt).

Входы CN0..CN30 имеют функцию уведомления об изменении уровня сигнала (отсюда и название входов – Change notification – CN – уведомление об изменении). Эта функция позволяет определить изменение уровня сигнала даже в спящем режиме микроконтроллера, генерируя соответствующее прерывание. Управление функцией осуществляется с помощью регистров CNEN1 и CNEN2, биты которых разрешают генерирование прерывание для каждого из входов CN0..CN30. Однако, для всех входов при любом изменении сигнала генерируется одно общее прерывание (_CNInterrupt), разрешение либо запрещение которого настраивается с помощью бита CNIE регистра IEC1.

### 1.3 Программирование прерываний

В компиляторе MPLAB C30 предусмотрен механизм работы с прерываниями посредством макросов, определенных в заголовочных файлах. В частности, биты разрешения прерываний определены как в структурах соответствующих регистров, так и в виде макросов. Например, для разрешения/запрета прерывания Таймера1 используется макрос _T1IE, для INT3 используется _INT3IE и т. д. Так, разрешить прерывание Таймера 1 в программе на Си можно с помощью оператора

```
_T1IE = 1;
```

либо посредством установки бита TON регистра T1CON:

```
T1CONbits.TON = 1;
```

Каждое прерывание, используемое в программе, должно иметь соответствующую функцию-обработчик, прототип которой в MPLAB C30 для можно представить следующим образом:

```
void __attribute__ ((__interrupt__)) isr0(void);
```

Из объявления прототипа функции-обработчика прерывания isr0 видно, что она не принимает никаких параметров и не возвращает никаких значений. При вызове функции-обработчика компилятор автоматически сохраняет в стеке все рабочие регистры, а также регистр состояния процессора SR. Остальные переменные можно сохранить, перечислив их в поле атрибутов. Например, для того чтобы компилятор автоматически сохранял и восстанавливал переменные var1 и var2, можно использовать такой прототип функции-обработчика:

```
void __attribute__((__interrupt__(__save__(var1, var2)))) isr0(void);
```

Название функции-обработчика прерывания начинается с двойного символа подчёркивания и имеют в своём названии обозначение источника прерывания и слово «Interrupt». Таким образом, объявление функции-обработчика прерывания таймера T1 будет иметь вид:

```
void __attribute__ ((__interrupt__)) __T1Interrupt(void);
```

А объявление функции-обработчика прерывания модуля АЦП1 будет следующим:

```
void __attribute__ ((__interrupt__)) __ADC1Interrupt(void);
```

## 2 Электрическая принципиальная схема к лабораторной работе

На рис. 2.1 приведена электрическая принципиальная схема к лабораторной работе.

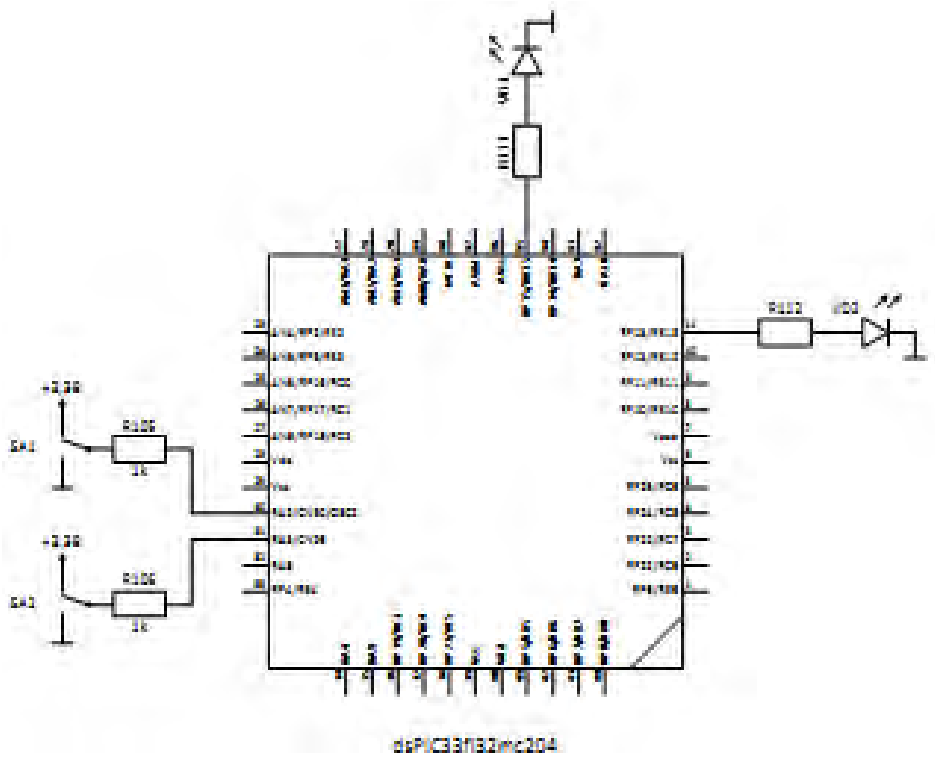


Рис. 2.1. Электрическая принципиальная схема к лабораторной работе

В схеме два переключателя SA1 и SA2 подключены к выводам RA2 (CN30) и RA3 (CN29) микроконтроллера. Два светодиода VD1 и VD2 подключены к выводам RB15 и RB13 микроконтроллера соответственно.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую при любом изменении состояний переключателей SA1, SA2 изменять состояние светодиодов VD1, VD2.

Листинг программы для решения задачи:

```
#include <P33FJ32MC204.h>
_FOSC(OSCIOFNC_ON &
POSCMD_NONE)                          // отключение дополнительной
                                      // функции порта RA2 – выход
                                      // тактирующего сигнала
                                      // внутреннего генератора

char ison = 0;       // хранение текущего состояния светодиодов
void __attribute__ ((__interrupt__)) _CNInterrupt()
{
      IFS1bits.CNIF = 0;              // Сброс флага прерывания
      if (ison)
      {
            LATBbits.LATB15       = 0;
            LATBbits.LATB13       = 0;
            ison = 0;
      }
      else
      {
            LATBbits.LATB15 =
            1; LATBbits.LATB13
            = 1; ison = 1;
      }
}
int main()
{
      TRISBbits.TRIS
      B15               = 0;        // настройка порта RB15 на выход
      TRISBbits.TRIS
      B13               = 0;        // настройка порта RB13 на выход
      TRISAbits.TRIS
      A2                = 1;        // настройка порта RA2 на вход
      TRISAbits.TRIS    = 1;        // настройка порта RA3 на вход
```

```
        A3
        CNEN2bits.CN2
        9IE             = 1;    // разрешение прерывания по
                                изменению
                                // входа CN29
        CNEN2bits.CN3
        0IE             = 1;    // разрешение прерывания по
                                изменению
                                // входа CN30
                                // разрешение прерываний по
        IEC1bits.CNIE   = 1;    событию
                                // Change Notification

        while (1)
        { }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, выполняющую следующие действия:

1. При изменении положения переключателя SA1 изменять состояние светодиода VD1, при изменении положения переключателя SA2 изменять состояние светодиода VD2.
2. Изменять состояние светодиода VD1 при появлении высокого уровня сигнала с любого из переключателей SA1, SA2.
3. Изменять состояние светодиода VD2 при появлении низкого уровня сигнала с любого из переключателей SA1, SA2.
4. Изменять состояние светодиода VD1 при появлении разных уровней сигнала с переключателей SA1, SA2.
5. Изменять состояние светодиода VD2 при появлении одинаковых уровней сигнала с переключателей SA1, SA2.

## 5 Контрольные вопросы

1. Дайте определение механизму прерывания.
2. Какие виды прерываний присутствуют в микроконтроллере dsPIC33?
3. Каким образом определяется порядок обработки прерываний?
4. Каким образом настраиваются прерывания?
5. Как происходит программирование прерываний на языке Си?

# Лабораторная работа 8

## Реализация временных функций в микропроцессорных системах управления

### Цель работы:

Изучить особенность программной и аппаратной реализации временных функций, режимы работы и порядок формирования таймеров микроконтроллера, реализацию временных функций с помощью языка программирования С.

### Порядок выполнения работы:

- Изучить теоретические вопросы, связанные с понятием машинного цикла, с принципом функционирования таймера-счётчика, системой прерываний.
- Изучить принципиальную электрическую схему к лабораторной работе.
- Разработать программу в соответствии с индивидуальным заданием.
- Отладить программу в среде MPLAB IDE.
- Загрузить программу в учебный стенд.
- Исследовать работу временных функций.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Реализация временных функций

Временные функции в микропроцессорных системах используются в случаях, когда необходимо выполнять какие либо действия периодически, либо производить измерение длительности какого либо события. Наибольшее распространение для реализации временных функций получили метод программных циклов и использование таймера/счётчика. Во всех случаях микроконтроллер оперирует не непосредственно временем, а количеством машинных циклов, выполненных за интересующий интервал времени. Машинный цикл микроконтроллера – это процедура выполнения ядром микроконтроллера одной инструкции за один машинный такт, равный двум периодам тактирующего генератора. Частота внутреннего тактирующего генератора микроконтроллера dsPIC33fj32mc204 (по умолчанию) равна 7.37МГц. Следовательно, машинный цикл длится  секунды.

### 1.2 Метод программных циклов

Метод программных циклов относится к программным способам реализации временной задержки и состоит в следующем. В некоторую переменную загружают число, которое затем в каждом проходе цикла уменьшается на 1. Так продолжается до тех пор, пока содержимое переменной не станет равной нолю, что интерпретируется программой как момент выхода из цикла, и, следовательно, истечения требуемого промежутка времени. Время задержки при этом определяется числом, загруженным в переменную-счетчик, и временем выполнения команд, образующих цикл.

Для организации временных задержек библиотека компилятора MPLAB C30 имеет специальную функцию для языка программирования C:

```
void __delay32(unsigned long cycles);
```

Недостатком программного способа реализации временной задержки является нерациональное использование ресурсов микроконтроллера: во время формирования задержки МК практически простаивает, так как не может решать никаких задач управления объектом. В то же время аппаратные средства микроконтроллера позволяют реализовать временные задержки на фоне основной программы работы.

### 1.3 Использование таймера/счётчика

Таймеры/счётчики (T/C) предназначены для подсчёта внешних событий, для получения программно-управляемых временных задержек и выполнения времязадающих функций микроконтроллера.

Микроконтроллеры семейства dsPIC33 имеют несколько 16-разрядных таймеров. Таймеры имеют обозначение Timer1, Timer2, Timer3 и т.д. С небольшими исключениями, все 16-битные таймеры имеют одинаковую функциональность, и разделены на 3 типа:

1. Таймер типа A (Timer1);
2. Таймер типа B (Timer2, Timer4, Timer6, Timer8);
3. Таймер типа C (Timer3, Timer5, Timer7, Timer9).

Таймеры типа B и C могут быть объединены для формирования 32-разрядного таймера.

Каждый из таймеров настраивается следующими регистрами, доступными для чтения и записи:

1. TMRx[4] – регистр счёта;

2. PRx – регистр периода;

3. TxCON – регистр конфигурации.

Также с каждым таймером ассоциированы следующие биты в регистре управления прерываниями:

1. TxIE – бит разрешения прерывания;

2. TxIF – бит статуса прерывания;

3. TxIP<2:0> – биты установки приоритета прерывания.

## 2 Электрическая принципиальная схема к лабораторной работе

На рис. 2.1 приведена электрическая принципиальная схема к лабораторной работе.

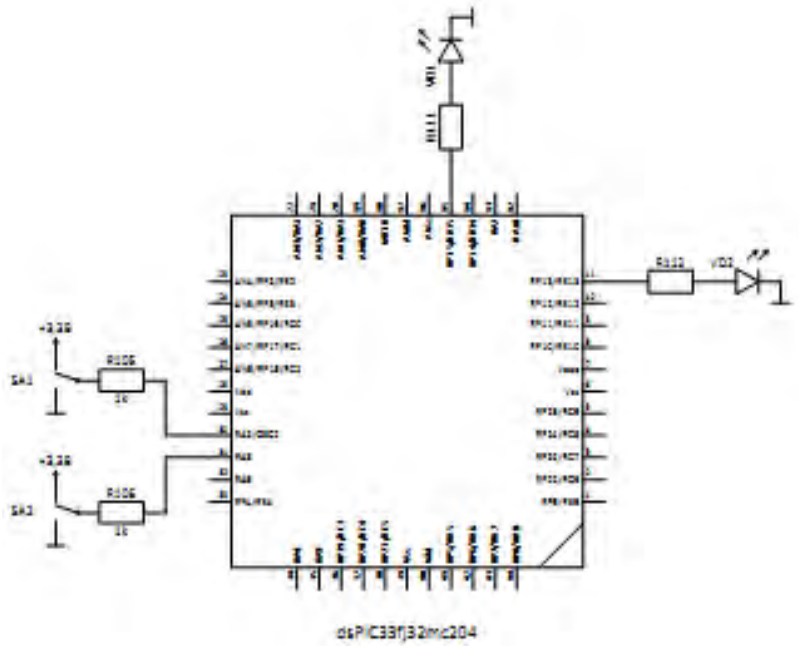


Рис. 2.1. Электрическая принципиальная схема к лабораторной работе

В схеме два дискретных входа оформлены в виде двух переключателей SA1 и SA2, подключенных к выводам RA2 и RA3 микроконтроллера. Два дискретных выхода оформлены в виде двух светодиодов VD1 и VD2, подключенных к выводам RB15 и RB13 микроконтроллера соответственно.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую мигать светодиодом VD1 с частотой 1Гц.

Метод программных циклов.
Листинг программы для решения задачи:

```
#include <P33FJ32MC204.h>
#define FOSC 7370000
#define FCY (FOSC / 2)
_FOSCSEL(FNOSC_FRC)     // настройка работы микроконтроллера
                        // от внутреннего тактового генератора
void main()
{
    TRISBbits.TRISB15 = 0;          // Выход  VD1  (RB15)
    while (1)
    {
        __delay32(FCY);
        LATBbits.LATB1
        5 = 1;
        __delay32(FCY);
        LATBbits.LATB1
        5 = 0;
    }
}
```

Использование таймера/счётчика.
*Анализ задачи:* Таймер/счётчик T1 настроен таким образом, что прерывания генерируются с частотой 1 кГц. В функции прерывания происходит увеличение значения переменной _ms на 1, таким образом, переменная _ms содержит количество миллисекунд, прошедших с момента запуска программы. В основном цикле происходит анализ значения переменной _ms, и в зависимости от этого, происходит управление светодиодом по заданному алгоритму.

Листинг программы для решения задачи:

```
#include
<P33FJ32MC204.h>
                                // настройка работы
_FOSCSEL(FNOSC_FRC)             микроконтроллера
                                // от внутреннего тактового
                                генератора
// Инициализация таймера
T1
```

```
void Init_Timer1()
{
        T1CON = 0;                  // сброс таймера
        IFS0bits.T1IF =
        0;                          // сброс флага прерывания таймера
        IEC0bits.T1IE =             // разрешение прерывания от
        1;                          таймера
        TMR1                        // обнуление текущего значения
        =       0x0000;             таймера
        PR1 =  0x0E65;              // задание периода таймера
                                = 1;  // разрешение работы таймера и его
        T1CONbits.TON  запуск
}
int _ms = 0;
// Прерывание таймера T1 по
совпадению void
__attribute__((interrupt))
_T1Interrupt()
{
        _ms++;
                                    // Сброс флага прерывания
        IFS0bits.T1IF = 0;          таймера
        TMR1       = 0;             // Перезапуск таймера
}
void main()
{
        Init_Timer1();
        TRISBbits.TRISB15 = 0;                  // Выход     VD1   (RB15)
        while (1)
        {
                if (_ms < 1000)
                {
                        LATBbits.LATB15 = 0;
                }
                else if (_ms < 2000)
                {
                        LATBbits.LATB15 = 1;
                }
                else
                {
                        _ms = 0;
                }
        }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. При включении тумблера SA1 мигать светодиодом VD1 с частотой 1Гц.
2. При включении тумблеров SA1 и SA2 одновременно мигать светодиодами VD1 и VD2 попеременно с частотой 1Гц.
3. При включении тумблера SA1 зажечь светодиод VD1 на 2 секунды.
4. При включении тумблера SA1 управлять светодиодом VD1 по следующему алгоритму: время включения – 2 с, время отключения – 1 с.
5. При любом изменении тумблера SA1 управлять светодиодом VD1 по следующему алгоритму: время включения – 1 с, время отключения – 2 с.

## 5. Контрольные вопросы

1. Какие существуют способы формирования временных задержек в микроконтроллерных системах?
2. Как формируется временная задержка методом программных циклов?
3. Как формируется временная задержка с использованием таймера/счётчика?
4. Каким образом можно определить длительность импульса?
5. Каким образом можно определить частоту сигнала?

## Лабораторная работа 9

## Исследование устройства динамической индикации

**Цель работы:**

Изучить алгоритм, принцип работы и схему электрическую принципиальную устройства динамической индикации. Разработать и отладить программу вывода информации на устройство динамической индикации.

**Порядок выполнения работы:**

- Изучить теоретические вопросы, связанные с устройствами статической и динамической индикации.
- Изучить принципиальную электрическую схему к лабораторной работе.
- Разработать программу в соответствии с индивидуальным заданием.
- Отладить программу в среде MPLAB IDE.
- Загрузить программу в учебный стенд.
- Исследовать работу устройства динамической индикации.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Устройства цифровой индикации

Для отображения цифровой индикации большое распространение получили светодиодные семисегментные индикаторы (рис. 1.1). Сегменты индикатора расположены в виде восьмёрки и обозначены латинскими буквами алфавита. Засвечивая группы сегментов, можно получить все цифры и некоторые символы.

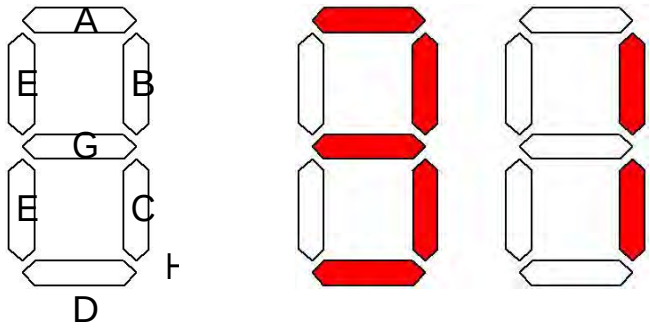



Рис. 1.1. Светодиодный семисегментный индикатор

Конструктивно индикаторы оформляют в виде светодиодных модулей с общим катодом или с общим анодом (рис. 1.2).

Рис. 1.2. Электрические принципиальные схемы семисегментных индикаторов с общим анодом (а) и с общим катодом (б)

При построении систем отображения информации различают два подхода: статическая и динамическая индикация.

Статическая индикация подразумевает постоянную засветку каждого из используемых индикаторов в любой момент времени. Таким образом, каждый индикатор подключён через свой регистр-защёлку к шине данных (рис. 1.3), что в свою очередь ведёт к большим аппаратным затратам.

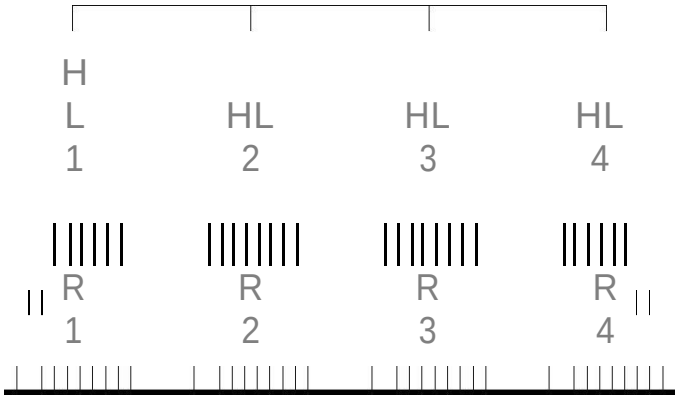



Рис. 1.3. Устройство статической индикации

Сущность динамической индикации состоит в поочерёдном циклическом подключении каждого индикатора к источнику данных (рис. 1.4). Таким образом, аппаратные затраты существенно снижаются. При реализации устройства динамической индикации с одной стороны каждому из индикаторов необходимо обеспечить достаточное время свечения для того, чтобы не уменьшалась яркость свечения индикаторов, а с другой стороны необходимо обеспечить достаточно быстрое переключение индикаторов, чтобы не было заметно мерцание.

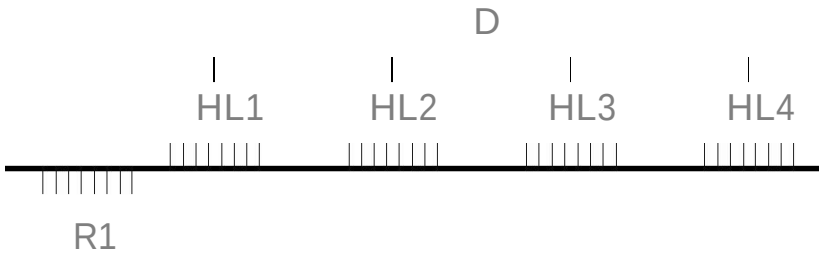



Рис. 1.4. Устройство динамической индикации

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая принципиальная схема к лабораторной работе.

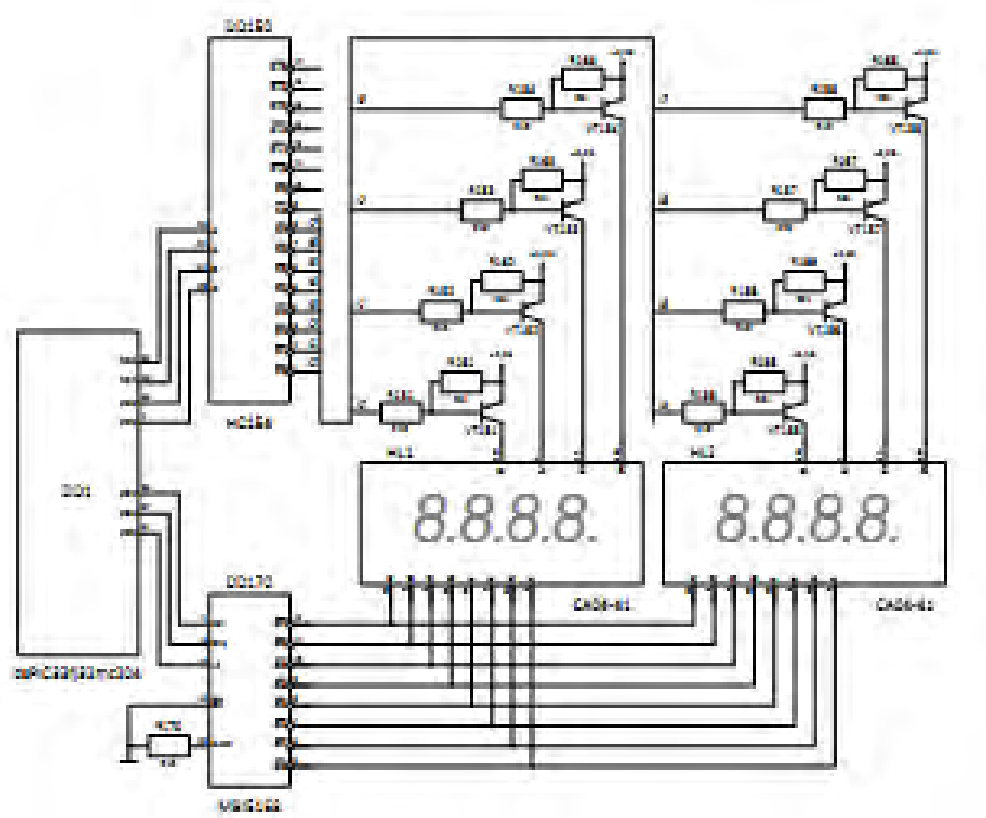



Рис. 2.1. Электрическая принципиальная схема к лабораторной работе

В качестве символьного светодиодного индикатора LED используется 2 индикатора CA04-41. Индикаторы представляют собой модули по четыре семисегментных светодиодных индикатора с общим анодом.

Для фиксации кода семисегментного символа служит восьмиканальный светодиодный драйвер DD170 MBI5168, представляющий собой сдвиговый регистр вывода, выходные каналы которого выполнены в виде стабилизатора тока для светодиодов. Сигнал тактирования CLK светодиодного драйвера подключен к линии RC4, сигнал данных – к RC3, сигнал защёлки – к линии RB5 порта микроконтроллера.

Для выбора активного индикатора используется дешифратор DD150 HC154. На вход дешифратора подаются сигналы A, B, C, D с линий портов RB6, RB7, RB8, RB9. Непосредственно на индикатор сигнал подаётся через усилитель тока, собранный на транзисторах VT151..VT158.

## 3 **Пример выполнения работы**

*Задача:* Разработать программу для учебного стенда, отображающую на LED количество секунд, прошедших после запуска программы.

*Анализ задачи:* Для хранения символов индикации используется массив _data[8], i-тый элемент которого представляет собой код символа, отображаемого на i-той позиции индикатора LED. Для хранения активного в данный момент времени индикатора используется переменная _ind. Переключение активного индикатора происходит по прерыванию таймера T1, настроенного таким образом, что прерывания генерируются с частотой 1 кГц. Количество секунд, прошедших с момента запуска программы, хранится в переменной seconds. В основном цикле программы происходит непрерывное увеличение значения переменной seconds на 1 с паузой 1 секунда.

Листинг программы:

```
#include <P33FJ32MC204.h>

#define FOSC
7370000 #define
FCY (FOSC / 2)

// объявление кодов сегментов
#define SEG_A          0x08          //      --- SEG_A ---
#define SEG_B          0x01          //      |            |
#define SEG_C          0x20          //   SEG_F   SEG_B
#define SEG_D          0x04          //      |            |
#define SEG_E          0x80          //      --- SEG_G ---
#define SEG_F          0x10          //      |            |
#define SEG_G          0x40          //   SEG_E   SEG_C
#define SEG_DP         0x02          //      |            |

                                     //      --- SEG_D ---   SEG_DP

// объявление кодов цифр
                       SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N0             SEG_F
#define N1             SEG_B + SEG_C
#define N2             SEG_A + SEG_B + SEG_G + SEG_E + SEG_D
#define N3             SEG_A + SEG_B + SEG_G + SEG_C + SEG_D
#define N4             SEG_F + SEG_G + SEG_B + SEG_C
#define N5             SEG_A + SEG_F + SEG_G + SEG_C + SEG_D
                       SEG_A + SEG_F + SEG_G + SEG_C + SEG_D +
#define N6             SEG_E
```

```
                    SEG_A + SEG_B +
#define N7          SEG_C
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N8          SEG_F + SEG_G
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_F +
#define N9          SEG_G

#define DIP         SEG_DP
#define
MINUS               SEG_G

                    0x0
#define OFF         0

// массив кодов цифр

char DIGITS[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};
_FOSCSEL(FNOSC_FRC) // настройка работы
микроконтроллера
                                    // от внутреннего тактового генератора
// Инициализация таймера
T1
void Init_Timer1()
{
        T1CON = 0;                  // сброс таймера
        IFS0bits.T1IF = 0;          // сброс флага прерывания таймера
                                    // разрешение прерывания от
        IEC0bits.T1IE = 1;          таймера
        TMR1                        // обнуление текущего значения
        =       0x0000;             таймера
        PR1 =  0x0E65;              // задание периода таймера
                                    // разрешение работы таймера и его
        T1CONbits.TON = 1;          запуск
}

char _ind = 0;      // номер активного индикатора

char _data[8];      // буфер индикации - массив кодов символов

// Отправление данных в регистр
MBI5168 void Ind_Send(char digit)

{
        char c;
```

```
        for (c = 0; c < 8; c++)
        {
            // установка требуемого логического уровня
            // на последовательном входе SDI драйвера
            MBI5168 if ((digit & (1 << c)) != 0)
            {
                LATCbits.LATC3 = 1;
            }
            else
            {
                LATCbits.LATC3 = 0;
            }
            // формирование синхроимпульса на входе SCK

            LATCbits.LATC
            4       =       1;
            LATCbits.LATC
            4 = 0;
        }
        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
LATBbits.LATB5 = 0;
    }

    // Инициализация линий портов
    void Ind_Init()
    {
                                                        (RB
        TRISBbits.TRISB6 = 0;           // Выход   A    6)
                                                        (RB
        TRISBbits.TRISB7 = 0;           // Выход   B    7)
                                                        (RB
        TRISBbits.TRISB8 = 0;           // Выход   C    8)
                                                        (RB
        TRISBbits.TRISB9 = 0;           // Выход   D    9)
                                                        (RC
        TRISCbits.TRISC3 = 0;           // Выход   SDI  3)
                                                SC      (RC
        TRISCbits.TRISC4 = 0;           // Выход   K    4)
                                                        (RB
        TRISBbits.TRISB5 = 0;           // Выход   LE   5)
    }

    // Разбитие числа на цифры
    void Ind_Show(unsigned int a, unsigned
```

```
int b)
{
        _data[3] = DIGITS[a % 10]; a /=
        10;
        _data[2] = DIGITS[a % 10]; a /=
        10;
        _data[1] = DIGITS[a % 10]; a /=
        10;
        _data[0] = DIGITS[a % 10];

        _data[7] = DIGITS[b % 10]; b /=
        10; _data[6] = DIGITS[b % 10];
        b /= 10; _data[5] = DIGITS[b %
        10];  b  /=  10;  _data[4]  =
        DIGITS[b % 10];
}
// Прерывание таймера T1 по совпадению
void __attribute__((interrupt,no_auto_psv)) _T1Interrupt()
{
        Ind_Send(OFF);          // Отключение индикатора
        LATB &= ~(0x07 << 6);// Переключение на следующий
        LATB |= (_ind << 6);    // индикатор
        Ind_Send(_data[_ind]);  // Отправка кода цифры
        _ind++;                 // переход к след. индикатору
        if (_ind == 8)
        {
                _ind = 0;
        }
IFS0bits.T1IF = 0;// Сброс флага прерывания таймера
        TMR1    = 0;            // Перезапуск таймера
}
void main()
{
        Init_Timer1();                  // Инициализация таймера
        Ind_Init();                     // Инициализация индикации
        Ind_Show(0, 0);                 // Обнуление индикации
        unsigned int seconds = 0;
        while (1)
        {
                Ind_Show(0, seconds);
                __delay32(FCY);         // пауза 1 сек
                seconds++;
        }

}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Отобразить на LED количество включённых тумблеров SA1..SA10 в десятичной форме.
2. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти их сумму и результат вывести на LED в десятичной форме.
3. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти их разность и результат вывести на LED в десятичной форме.
4. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти их произведение и результат вывести на LED в десятичной форме.
5. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти результат поразрядного логического ИЛИ и результат вывести на LED в двоичной форме.
6. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти результат поразрядного логического И и результат вывести на LED в двоичной форме.
7. Рассматривая состояние дискретных датчиков SA3..SA10 как два четырёхразрядных двоичных числа, найти результат поразрядного логического ИСКЛЮЧАЮЩЕГО ИЛИ и результат вывести на LED в двоичной форме.
8. Отобразить на LED поразрядно состояние тумблеров SA3..SA10, отображая включённый SA единицей, а отключённый SA отключённым символом.

## 5 Контрольные вопросы

1. Как организуется вывод информации в микроконтроллерных системах?
2. Как устроен светодиодный семисегментный индикатор.
3. Расскажите принцип построения и алгоритм работы статической индикации.
4. Расскажите принцип построения и алгоритм работы динамической индикации.

# Лабораторная работа 10

## Исследование ввода информации при помощи клавиатуры

**Цель работы:**

Изучить алгоритм, принцип работы и схему электрическую принципиальную матричной клавиатуры. Разработать и отладить программу ввода информации в микроконтроллер с помощью клавиатуры.

**Порядок выполнения работы:**

1. Изучить теоретические вопросы, связанные с вводом информации с помощью клавиатуры.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу устройства динамической индикации.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Устройство матричной клавиатуры

Клавиатура предназначена для ввода информации в микроконтроллерное устройство. Фактически, клавиатура представляет собой набор дискретных переключателей. В случае большого количества таких переключателей требуется использовать большое количество линий ввода.

Одним из наиболее распространённых способов уменьшения требуемых линий для подключения клавиатуры является организация клавиатуры по принципу матричного шифратора, в узлах которого расположены переключатели (рис. 1.1).

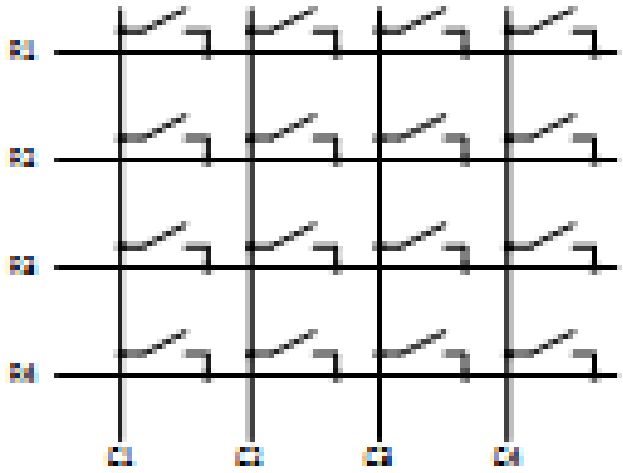


Рис. 1.1. Матричная организация клавиатуры

Кнопки включены таким образом, что при нажатии кнопка замыкает строку на столбец. Часть линий матрицы используется в качестве сканирующих (строки), а часть в качестве считывающих (столбцы). Количество кнопок, подключенных таким образом, определяется как количество сканирующих линий, умноженное на количество считывающих. Отсюда следует, что использование матричной клавиатуры для случая, когда кнопок меньше или равно четырем, не имеет смысла, так как понадобятся те же четыре линии, а схема и алгоритм опроса клавиш усложняются.

### 1.2 Алгоритм работы матричной клавиатуры

Работает матричная клавиатура следующим образом. На одну их линий R1..R4 подаётся сигнал, т.е. происходит сканирование одного из рядов клавиатуры. Если ни одна из клавиш не нажата, то сигнал на линиях C1..C4 отсутствует. Если клавиша на сканирующем ряду нажата, то на соответствующей линии C1..C4 появляется сигнал. Таким образом, зная какой ряд в данный момент сканируется и на какой из линий C1..C4 появился сигнал, можно определить, какая клавиша зажата в данный момент.

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая принципиальная схема к лабораторной работе.

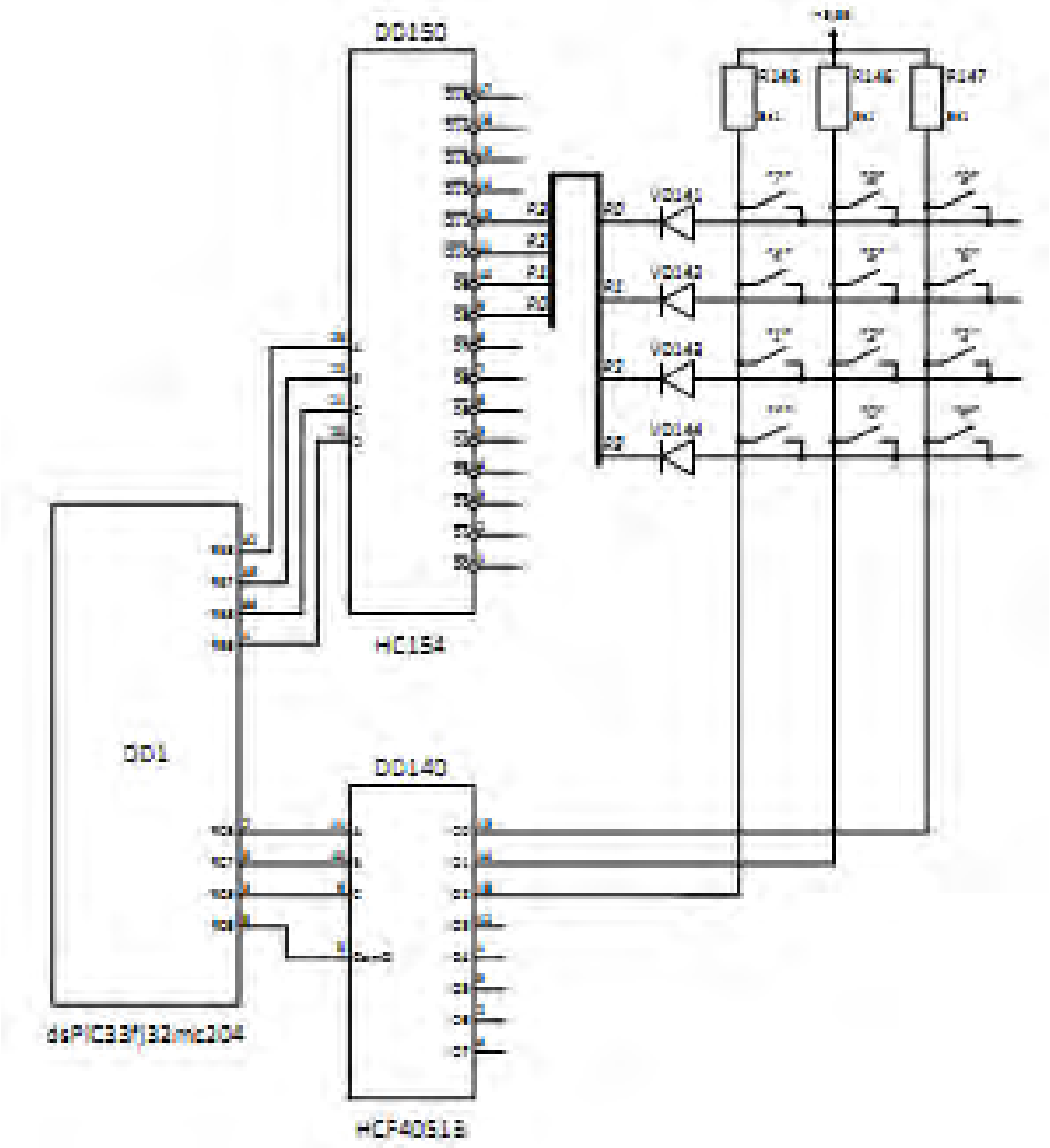



Рис. 2.1. Электрическая принципиальная схема к лабораторной работе

Для выбора строки матричной клавиатуры используются выводы D8..D11 дешифратора DD150 HC154. На вход дешифратора подаются сигналы A, B, C, D с линий портов RB6, RB7, RB8, RB9.

Для сканирования столбцов используется восьмиканальный мультиплексор DD140 HCF4051B. Выбор активного входа мультиплексора происходит сигналами A, B, C с линий RC6, RC7, RC8 микроконтроллера. Выход мультиплексора подаётся на вход RC9 микроконтроллера.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, отображающую на LED строку и столбец нажатой клавиши матричной клавиатуры.

*Анализ задачи:* Для хранения символов индикации используется массив _ind[8], каждый элемент которого представляет собой код символа, отображаемого на соответствующей позиции индикатора LED. Для хранения активного в данный момент вывода дешифратора используется переменная _i. Переключение активного вывода дешифратора происходит по прерыванию таймера T1, настроенного таким образом, что прерывания генерируются с частотой 1 кГц. В случаях, когда активный вывод дешифратора соединён со строкой матричной клавиатуры, запускается процедура опроса состояний столбцов. Состояние клавиш сохраняется в двумерном массиве _keys[4][3], первый индекс которого является номером строки, второй – номером столбца.

Листинг программы:

```
#include <P33FJ32MC204.h>
#define FOSC
7370000 #define
FCY (FOSC / 2)
// объявление кодов сегментов
#define SEG_A           0x08        //      --- SEG_A ---
#define SEG_B           0x01        //      |         |
#define SEG_C           0x20        //   SEG_F   SEG_B
#define SEG_D           0x04        //      |         |
#define SEG_E           0x80        //      --- SEG_G ---
#define SEG_F           0x10        //      |         |
#define SEG_G           0x40        //   SEG_E   SEG_C
#define SEG_DP          0x02        //      |         |

                                    //       --- SEG_D ---   SEG_DP

// объявление кодов цифр
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N0          SEG_F
#define N1          SEG_B + SEG_C
#define N2          SEG_A + SEG_B + SEG_G + SEG_E + SEG_D
#define N3          SEG_A + SEG_B + SEG_G + SEG_C + SEG_D
#define N4          SEG_F + SEG_G + SEG_B + SEG_C
#define N5          SEG_A + SEG_F + SEG_G + SEG_C + SEG_D
                    SEG_A + SEG_F + SEG_G + SEG_C + SEG_D +
#define N6          SEG_E
                    SEG_A + SEG_B +
#define N7          SEG_C
```

```
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N8          SEG_F + SEG_G
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_F +
#define N9          SEG_G

#define DIP         SEG_DP

#define MINUS       SEG_G

#define OFF         0x00
// массив кодов цифр
char DIGITS[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};

                                // настройка работы
_FOSCSEL(FNOSC_FRC)             микроконтроллера
                                // от внутреннего тактового
                                генератора
                                // номер активного вывода
char _i = 0;                    дешифратора
                                // буфер индикации - массив кодов
char _ind[8];                   символов
char _keys[4][3];               // буфер клавиатуры
// Инициализация таймера
T1
void
Init_Timer1()
{
                                // сброс
        T1CON       = 0;        таймера
        IFS0bits.T1
        IF          = 0;        // сброс флага прерывания таймера
        IEC0bits.T1IE = 1;      // разрешение прерывания от таймера
                                // обнуление текущего значения
        TMR1        = 0x0000;   таймера
        PR1         = 0x0E65;   // задание периода таймера
                                // разрешение работы таймера и его
        T1CONbits.TON = 1;      запуск
}
// Инициализация линий
портов
void Ind_Init()
{
        TRISBbits.TRISB6 =
        0;                              // Выход    A     (RB6)
        TRISBbits.TRISB7 =              // Выход    B     (RB7)
```

```
        0;
        TRISBbits.TRISB8 =
        0;                              // Выход    C       (RB8)
        TRISBbits.TRISB9 =
        0;                              // Выход    D       (RB9)
        TRISCbits.TRISC3 =
        0;                              // Выход    SDI     (RC3)
        TRISCbits.TRISC4 =                          SC
        0;                              // Выход    K       (RC4)
        TRISBbits.TRISB5 =
        0;                              // Выход    LE      (RB5)
}
// Отправление данных в регистр
MBI5168 void Ind_Send(char digit)
{
        char c;
        for (c = 0; c < 8; c++)
        {
                1. установка требуемого логического уровня
                2. на последовательном входе SDI драйвера MBI5168
                if ((digit & (1 << c)) != 0)
                {
                        LATCbits.LATC3 = 1;
                }
                else
                {
                        LATCbits.LATC3 = 0;
                }
                // формирование синхроимпульса на входе SCK
                LATCbits.LATC4 = 1;
                LATCbits.LATC4 = 0;
        }
        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
        LATBbits.LATB5 = 0;
}
void Ind_Show(char a, char b)
{
        _ind[3] = DIGITS[a];
        _ind[7] = DIGITS[b];
}
// Инициализация линий
портов
void Kbd_Init()
{
```

```
                                                    (RC
        TRISCbits.TRISC6 = 0;       // Out      INA 6)
                                                    (RC
        TRISCbits.TRISC7 = 0;       // Out      INB 7)
                                                    (RC
        TRISCbits.TRISC8 = 0;       // Out      INC 8)
                                                    (RC
        TRISCbits.TRISC9 = 1;       // In       D   9)
}
void Kbd_ReadCol(char row)
{
        char column;
        for (column = 0; column < 3; column++)
        {
                LATC &= ~(0x07 <<
                6);                         // Выбор входа
                LATC |= (column <<
                6);

                __delay32(100L);            // Пауза для установки сигнала

                // чтение столбца

                if (PORTCbits.RC9)
                {
                        _keys[row][column] = 0;
                }
                else
                {
                        _keys[row][column] = 1;
                }
        }
}
// Прерывание таймера T1 по совпадению
void __attribute__((interrupt,no_auto_psv)) _T1Interrupt()
{
        Ind_Send(OFF);                      // Отключение индикатора
        LATB &= ~(0x0F <<
        6);                                 // Переключение на следующий
        LATB |= (_i << 6);                  // индикатор


        if (_i < 8)
        {
                Ind_Send(_ind[_i]           // Отправка кода цифры
```

```
            );
        }
        else if (_i < 12)
        {
            Kbd_ReadCol(_i -
            8);                         // Сканирование столбца
        }
                                    // переключение активного вывода
        _i++;                       индикатора
        if (_i ==
        12)
        {
            _i = 0;
        }

                                    // Сброс флага прерывания
        IFS0bits.T1IF = 0;          таймера
        TMR1    = 0;                // Перезапуск таймера
    }


    void main()
    {
        Kbd_Init()
        ;
        Ind_Init();                 // Инициализация индикации

Init_Timer1();      // Инициализация таймера
        while (1)
        {
            char row, column;
            for (row = 0; row < 4; row++)
            {
                for (column = 0; column < 3; column++)
                {
                    if (_keys[row][column] != 0)
                    {
                        Ind_Show(row, column);
                    }
                }
            }
        }
    }
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать сумму введённых чисел и вывести результат на LED.
2. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать разность введённых чисел и вывести результат на LED.
3. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать произведение введённых чисел и вывести результат на LED.
4. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать остаток от деления введённых чисел и вывести результат на LED.
5. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать наибольший общий делитель введённых чисел и вывести результат на LED.
6. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать наименьшее общее кратное введённых чисел и вывести результат на LED.
7. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода вывести максимальное из чисел на VD1..VD10 в двоичном коде.
8. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LED. Клавишу «#» использовать как окончание ввода числа. После ввода вывести минимальное из чисел на VD1..VD10 в двоичном коде.

## 5 Контрольные вопросы

1. Для чего используется клавиатура в микропроцессорных системах?
2. Какое преимущество даёт использование матричной клавиатуры?
3. Расскажите алгоритм опроса клавиш матричной клавиатуры.
4. Какие недостатки присущи матричной клавиатуре?

# Лабораторная работа 11

## Исследование устройства матричной жидкокристаллической индикации

### Цель работы:

Изучить алгоритм, принцип работы и схему электрическую принципиальную подключения матричной жидкокристаллической индикации. Разработать и отладить программу вывода информации на ЖКИ.

### Порядок выполнения работы:

- Изучить теоретические вопросы, связанные с работой матричного жидкокристаллического индикатора.
- Изучить принципиальную электрическую схему к лабораторной работе.
- Разработать программу в соответствии с индивидуальным заданием.
- Отладить программу в среде MPLAB IDE.
- Загрузить программу в учебный стенд.
- Исследовать работу устройства жидкокристаллической индикации.
- Оформить отчёт по лабораторной работе.
- Ответить на контрольные вопросы.

### 1. Краткие теоретические сведения

### 1.1 Устройство и принцип работы жидкокристаллического индикатора

В настоящее время в микропроцессорных системах для отображения широко используют жидкокристаллические индикаторы (ЖКИ). Условно все ЖКИ можно разделить на две категории: символьные, или знакосинтезирующие, и графические. Графические индикаторы представляют собой матрицу из m строк и n столбцов, на пересечении которых находятся пиксели. Пиксель представляет собой неделимый объект прямоугольной или круглой формы, обладающий определённым цветом; пиксель – наименьшая единица растрового изображения. Если на определенный столбец и строку подать электрический сигнал, то пиксель на их пересечении изменит свой цвет. Подавая группу сигналов на столбцы и строки можно формировать по точкам произвольное графическое изображение. Так работает графический ЖКИ. В символьном же ЖКИ матрица пикселей разбита на подматрицы, каждая подматрица предназначена для формирования одного символа: цифры, буквы или знака препинания. Как правило, для формирования одного символа используют матрицу из восьми строк и пяти столбцов. Символьные индикаторы бывают одно-, двух- и четырехстрочными.

В состав контроллера ЖКИ входят три вида памяти: CGROM, CGRAM, DDRAM. Когда микроконтроллер передает в контроллер ЖКИ коды символов, то они записываются в DDRAM (Display data RAM – ОЗУ кодов отображаемых символов), такую память называют видеопамятью или видеобуфером. Видеобуфер в символьных

индикаторах обычно содержит 80 ячеек памяти – больше, чем число знакомест дисплея. У двухстрочных индикаторов ячейки с адресами от 0x00 и до 0x27 отображаются на верхней строке дисплея, а ячейки с адресами 0x40 … 0x67 – на нижней строке.

Матрицы начертания символов хранятся в памяти знакогенератора. Память знакогенератора включает в себя CGROM (Character generator ROM – ПЗУ знакогенератора), в которую на заводе-изготовителе загружены начертания символов таблицы ASCII. Содержимое CGROM изменить нельзя. Для того, чтобы пользователь смог самостоятельно задать начертание нужных ему символов, в знакогенераторе имеется специальное ОЗУ – CGRAM (Character generator RAM). Под ячейки CGRAM отведены первые (младшие) 16 адресов таблицы кодов.

Для управления индикатором предназначены 11 линий — восемь для передачи данных (D0 - D7) и три линии управления. Линия RS служит для сообщения контроллеру индикатора о том, что именно передается по шине: команда или данные (RS = 1 — данные, RS = 0 — команда). По линии E передается строб-сигнал, сопровождающий запись или чтение данных: по переходу сигнала на линии E из 1 в 0 осуществляется запись данных во входной буфер микроконтроллера индикатора. Запись информации в ЖКИ происходит по спаду этого сигнала. Потенциал на управляющем выводе R/W (Read/Write) задает направление передачи информации, при R/W = 0 осуществляется запись в память индикатора, при R/W = 1 – чтение из нее. Еще три линии предназначены для подачи питающего напряжения (VDD, GND) и напряжения смещения, которое управляет контрастностью дисплея.

Буфер ввода/вывода индикатора может работать в восьми- и четырёхбитном режиме. В четырёхбитном режиме данные передаются тетрадами.

После приема информации контроллеру ЖКИ требуется некоторое время на выполнение команд, в это время управляющий контроллер не должен давать следующую команду или пересылать данные.

Команды и алгоритмы работы жидкокристаллических индикаторов различных производителей могут иметь существенные различия. Организация работы с индикатором будет рассмотрена на примере используемого в стенде ЖКИ DV-16236. Размер индикатора – 2 строки по 16 символов. Используемый драйвер – HD44780.

## 1.2 Команды контроллера ЖКИ

В таблице 1.1 приведены основные команды контроллера ЖКИ и время, необходимое для выполнения этих команд. Значения битов используемых при установке режимов работы индикатора приведены в таблице 1.2.

Для того чтобы можно было определить, когда ЖКИ закончит свои внутренние операции, контроллер ЖКИ содержит специальный флаг занятости – BUSY-флаг (BF). Если контроллер занят выполнением внутренних операций, то BF установлен (BF = 1), если же контроллер готов принять следующую команду, то BF сброшен (BF = 0). Более простой способ организации обмена заключается в том, что управляющий микроконтроллер, зная, сколько времени требуется ЖКИ на

обработку той или иной команды, после каждой передачи информации ждет соответствующее время.

Таблица 1.1

Коды команд контроллера ЖКИ

| *Инструкция* | *Код инструкции* | | | | | | | | | | *Описание* | *Время выполнения* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *RS* | *R/W* | *DB7* | *DB6* | *DB5* | *DB4* | *DB3* | *DB2* | *DB1* | *DB0* | | |
| Очистить дисплей | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Очистка дисплея и перевод курсора в начальную позицию | 1,52 мс |
| Возврат в начало | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | - | Перевод курсора в начальную позицию | 1,52 мс |
| Установка режима ввода | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | SH | Установка направления перемещения курсора и разрешение смещения отображаемой области | 37 мс |
| Вкл/выкл индикатора | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Установка битов вкл/выкл дисплея (D), курсора (C) и мерцания курсора (B) | 37 мс |
| Сдвиг курсора и индикатора | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | - | - | Установка битов перемещения курсора и сдвига дисплея без изменения DDRAM | 37 мс |
| | | | | | | | | | | | Установка интерфейса | |

Окончание таблицы 1.1

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Установка режимов | 0 | 0 | 0 | 0 | 1 | DL | N | F | - | - | (DL: 8/4-битный), количества строк (N) и шрифта (F) | 37 мс |
| Установка позиции курсора | 0 | 0 | 1 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Установка интерфейса позиции курсора (адреса DDRAM) | 37 мс |
| Запись данных в DDRAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Запись данных в DDRAM | 37 мс |

Таблица 1.2

Значение битов конфигурации

| *Значение бита* | *Описание* |
|---|---|
| I/D = 0 | Смещение курсора вправо при записи данных в DDRAM |
| I/D = 1 | Смещение курсора влево при записи данных в DDRAM |
| SH = 0 | Смещение отображаемой области отключено |
| SH = 1 | Смещение отображаемой области включено |
| D = 0 | Отключить дисплей |
| D =1 | Включить исплей |
| C = 0 | Отключить курсор |
| C = 1 | Включить курсор |
| B = 0 | Отключить мерцание курсора |
| B = 1 | Включить мерцание курсора |
| N = 0 | 1-строчный дисплей |
| N = 1 | 2-строчный дисплей |
| F = 0 | Размер символа 5*8 |
| F = 1 | Размер символа 5*10 |

### 1.3 Инициализация ЖКИ

Перед началом работы требуется произвести инициализацию ЖКИ. Алгоритм инициализации индикатора для четырёхбитного режима представлен на рисунке 1.2.

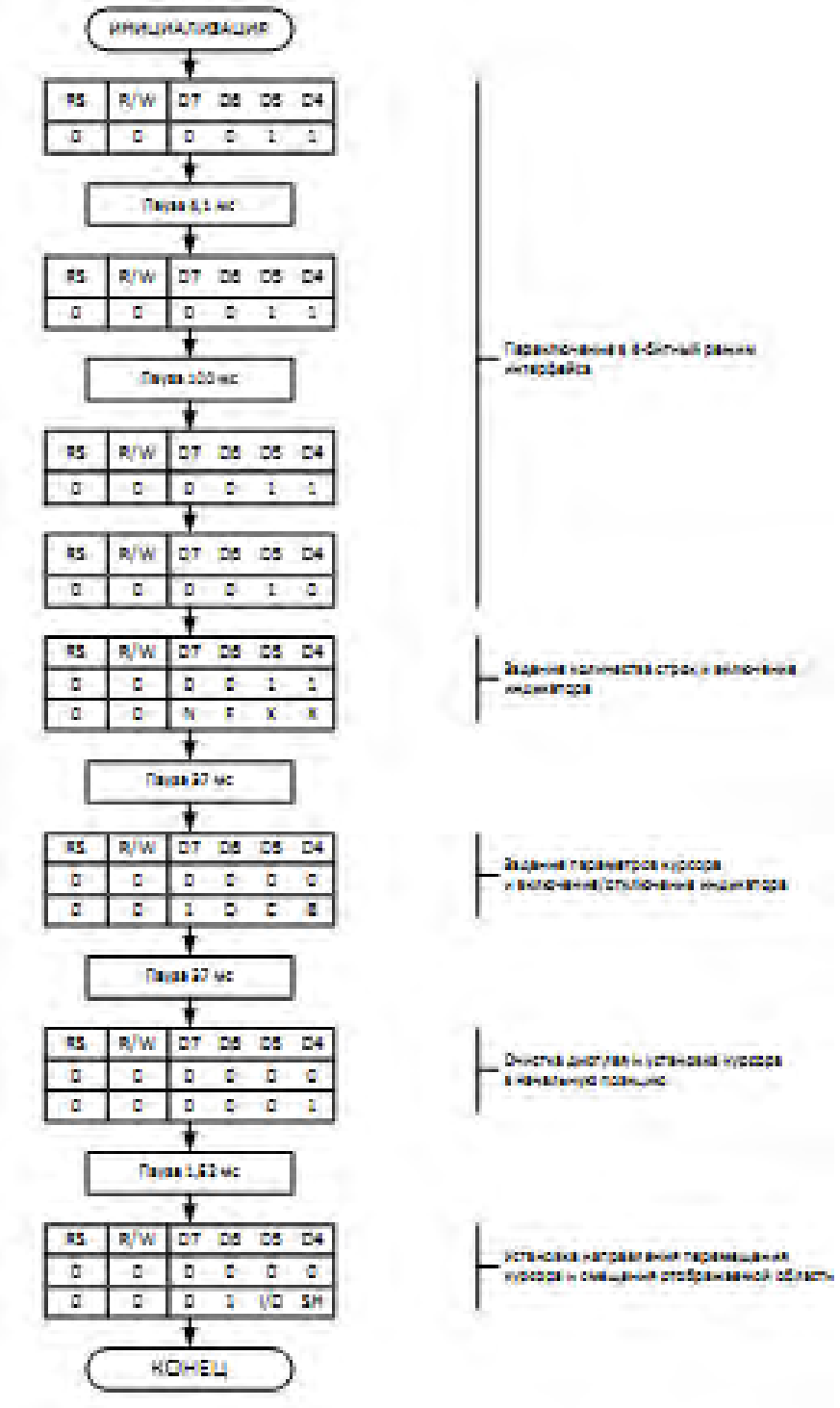



Рис. 1.2. Блок-схема алгоритма инициализации индикатора

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая схема подключения матричного жидкокристаллического индикатора к микроконтроллеру.

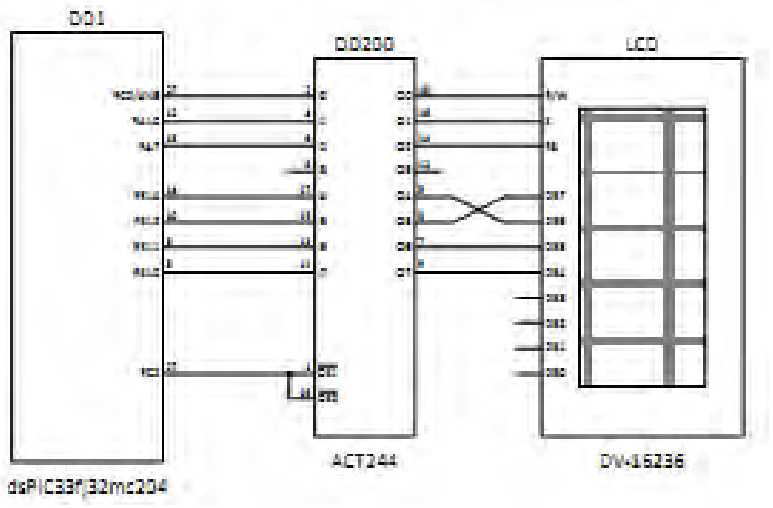



Рис. 2.1. Электрическая схема подключения матричного жидкокристаллического индикатора

В стенде используется матричный жидкокристаллический индикатор DV-16236 фирмы DataVision. Для согласования уровней напряжений микроконтроллера и жидкокристаллического индикатора, последний подключён к микроконтроллеру через

микросхему буфера. Активация буфера происходит при низком уровне сигнала на входах.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, отображающую на LCD строку с моделью используемого в стенде микроконтроллера.

*Анализ задачи:*

Программу для работы с ЖКИ следует организовать в виде функций, выполняющих определенные действия, причем более сложные функции могут включать в себя простейшие. В виде подпрограмм целесообразно реализовать такие функции, как процедура инициализации драйвера ЖКИ, операция отправки команды контроллеру и функция, записывающая данные в DDRAM.

Руководствуясь диаграммой передачи информации из документации на драйвер индикатора, определим последовательность действий при передаче информации в ЖКИ следующим образом: устанавливаем требуемое значение RS, на линию R/W подаем логический ноль, затем на линию E выводим

логическую единицу, после чего подаем на шину D значение передаваемого байта. Контроллер ЖКИ считает этот байт и состояние управляющих линий (RS, R/W) только после подачи на линию E логического ноля.

Временные задержки, требуемые для выполнения команд драйвером ЖКИ, реализуются методом программных циклов. Для задания времени паузы в миллисекундах определён макрос, пересчитывающий количество миллисекунд в число машинных циклов и вызывающий стандартную функцию реализации паузы.

Листинг программы:

```
#include <P33FJ32MC204.h>
#define FOSC
7370000 #define
FCY (FOSC / 2)
#define Delay_ms(d) (__delay32 (((d)) * ((FCY) / 1000uL)))
_FOSCSEL(FNOSC_FRC)     // настройка работы микроконтроллера
                                    // от внутреннего тактового
                                    генератора
#define LCD_EN          LATAbits.LATA10
#define LCD_RW          LATCbits.LATC2
#define LCD_RS          LATAbits.LATA7
#define LCD_DB7         LATBbits.LATB12
#define LCD_DB6         LATBbits.LATB14
#define LCD_DB5         LATBbits.LATB11
#define LCD_DB4         LATBbits.LATB10
// Подпрограмма отправки тетрады данных в
драйвер ЖКИ void LCD_set_half_byte(char x)
{
        if (x & (0x01 <<
                3))
                LCD_DB7
                = 1;
        else
                LCD_DB7 = 0;

        if (x & (0x01 <<
                2))
                LCD_DB6
                = 1;
        else

                LCD_DB6 = 0;
```

```
        if (x & (0x01 << 1))
                LCD_DB5 = 1;
        else
                LCD_DB5
                = 0;
        if (x & (0x01 <<
        0))
                LCD_DB4
                = 1;
        else
                LCD_DB4
                = 0;

        LCD_EN = 1;
        Delay_ms(1);
        LCD_EN = 0;
        Delay_ms(1);
}
// Инициализация используемых линий
порта
void LCD_InitPorts()
{
        TRISBbits.TRISB                 //                      (RB1
        1               = 0;            Out             OE      )
        LATBbits.LATB1          = 0;            // OE           Lo
        TRISAbits.TRISA                 //                      (RA1
        10              = 0;            Out             En      0)
        TRISCbits.TRISC                 //                      (RC2
        2               = 0;            Out             R/W     )
        TRISAbits.TRISA                 //                      (RA7
        7               = 0;            Out             RS      )
        TRISBbits.TRISB                 //                      (RB1
        12              = 0;            Out             DB7     2)
        TRISBbits.TRISB                 //                      (RB1
        14              = 0;            Out             DB6     4)
        TRISBbits.TRISB                 //                      (RB1
        11              = 0;            Out             DB5     1)
        TRISBbits.TRISB                 //                      (RB1
        10              = 0;            Out             DB4     0)
}

// Инициализация контроллера
ЖКИ void LCD_Init()
{
        LCD_RS = 0;
```

```
        LCD_RW = 0;

        LCD_EN = 0;
        Delay_ms(30);
        LCD_set_half_byte(0x03);
        Delay_ms(5);
        LCD_set_half_byte(0x03);
        Delay_ms(100);
        LCD_set_half_byte(0x03);
        LCD_set_half_byte(0x02);
```

- 4-битный интерфейс, 2 строки, размер символа - 5x8

```
        LCD_set_half_byte(0
        x02);
        LCD_set_half_byte(0
        x08); Delay_ms(37);
```

- Курсор выкл., мигание позиции выкл., индикатор вкл.

```
        LCD_set_half_byte(0
        x00);
        LCD_set_half_byte(0
        x0C); Delay_ms(37);
```

- Очитска дисплея

```
        LCD_set_half_byte(0x00);
        LCD_set_half_byte(0x01);
        Delay_ms(2);
        // Направление перемещения курсора - вправо
        LCD_set_half_byte(0x00);
        LCD_set_half_byte(0x06);
}
// Отправление данных в DDRAM
контроллера ЖКИ void LCD_send_char(char
c)
{
        LCD_RS = 1;
        LCD_set_half_byte(c >> 4);
        LCD_set_half_byte(c & 0x0f);
        LCD_RS = 0;
}
void LCD_WriteString(char str[])
{
        char i = 0;
        while (str[i] != 0)
        {
```

```
                LCD_send_char(str
                [i]); i++;
        }
}
void main()
{
        AD1PCFGL = 0xFFFF;          // Отключение АЦП
        LCD_InitPorts();
        LCD_Init();
        LCD_WriteString("dsPIC33fj32mc204\0");
        while (1)
        {
        }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать сумму введённых чисел и вывести результат на LCD.
2. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать разность введённых чисел и вывести результат на LCD.
3. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать произведение введённых чисел и вывести результат на LCD.
4. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать остаток от деления введённых чисел и вывести результат на LCD.
5. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать наибольший общий делитель введённых чисел и вывести результат на LCD.
6. Ввести последовательно два числа с клавиатуры, отображая вводимое число на LCD. Клавишу «#» использовать как окончание ввода числа. После ввода рассчитать наименьшее общее кратное введённых чисел и вывести результат на LCD.

## 5 Контрольные вопросы

1. Какой физический принцип заложен в сущности работы ЖКИ?
2. Какие используются режимы отображения ЖКИ?
3. Какое типичное время срабатывания ЖКИ?
4. На какие категории по принципу отображения информации делятся ЖКИ?
5. Как устроен ЖКИ?
6. Какие функции выполняет драйвер ЖКИ?
7. Каким образом происходит управление ЖКИ?

# Лабораторная работа 12

## Исследование средств ввода аналоговой информации в микроконтроллер

### Цель работы:

Изучить алгоритм и принцип настройки и работы аналого-цифрового преобразователя. Разработать и отладить программу для сбора аналоговой информации.

### Порядок выполнения работы:

1. Изучить теоретические вопросы, связанные с работой аналого-цифрового преобразователя.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу системы сбора аналоговой информации.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Аналого-цифровые преобразователи

Аналого-цифровые преобразователи (АЦП) предназначены для преобразования аналоговой величины в цифровой код. Другими словами, АЦП - это устройства, которые принимают аналоговые сигналы и генерируют соответствующие им цифровые значения.

Существуют различные схемы построения АЦП, отличающиеся по принципу оцифровки сигнала, быстродействию, помехоустойчивости и т.д.

В системах, где основным критерием является быстродействие, применяют АЦП параллельного преобразования. АЦП этого типа сложны в реализации. Для n-разрядного АЦП необходимо $2^{n-1}$ компараторов и параллельный делитель напряжения, который вырабатывает $2^{n-1}$ уровней квантования.

Для реализации систем с высокой помехоустойчивостью применяются интегрирующие АЦП. Такой АЦП состоит их двух преобразователей. Измеряемое напряжение преобразуется в длительность импульса, а потом длительность импульса преобразуется в цифровой код.

Довольно распространённой является схема построения АЦП, основанная на последовательном сравнении измеряемого сигнала с ½, ¼, ⅛ и т. п. от возможного его максимального значения.

## 1.2 АЦП микроконтроллера

Микроконтроллеры семейства dsPIC33 могут иметь до 32 аналоговых входов и до 2 модулей АЦП, каждый из которых имеет свой собственный набор регистров управления. В частности, dsPIC33fj32mc204 имеет 9 аналоговых входов, обозначаемых AN0..AN8, и 1 модуль АЦП (ADC0). Аналого-цифровой преобразователь микроконтроллера разработан на основе регистра последовательного приближения. Установка режимов работы и рабочих параметров преобразователя осуществляется с помощью регистров

AD1CON1…AD1CON3, AD1CHS, AD1PCFG и AD1PCFG.

АЦП может работать в 10- либо 12-битном режиме. В качестве опорного напряжения может использоваться как напряжение питания контроллера, так и напряжение, подаваемое на специальные входы $V_{REF+}$ и $V_{REF-}$. Результаты преобразования сохраняются в массиве 16-битных слов ADC1BUF0.. ADC1BUF8. После каждого полного цикла преобразования может генерироваться соответствующее прерывание.

Структурная схема АЦП микроконтроллера представлена на рисунке 1.1.

Входы аналоговых сигналов через специальный мультиплексор подключаются к усилителю выборки и хранения аналого-цифрового преобразователя. Аналоговый мультиплексор настраивается регистрами AD1CHS123 и AD1CHS0. В каждом из этих регистров присутствует два набора управляющих битов, функциональность которых идентична. Эти два набора позволяют задать две различных конфигурации подключения аналоговых входов к усилителям, варианты подключения при этом обозначают MUXA и MUXB. Модуль АЦП позволяет переключать эти варианты перед началом преобразования.

В преобразователе предусмотрен вариант автоматического сканирования нескольких каналов ввода, режимы автовыборки и автопреобразования, что значительно упрощает управление преобразователем и избавляет разработчика от трудоёмких расчётов временных характеристик преобразования.

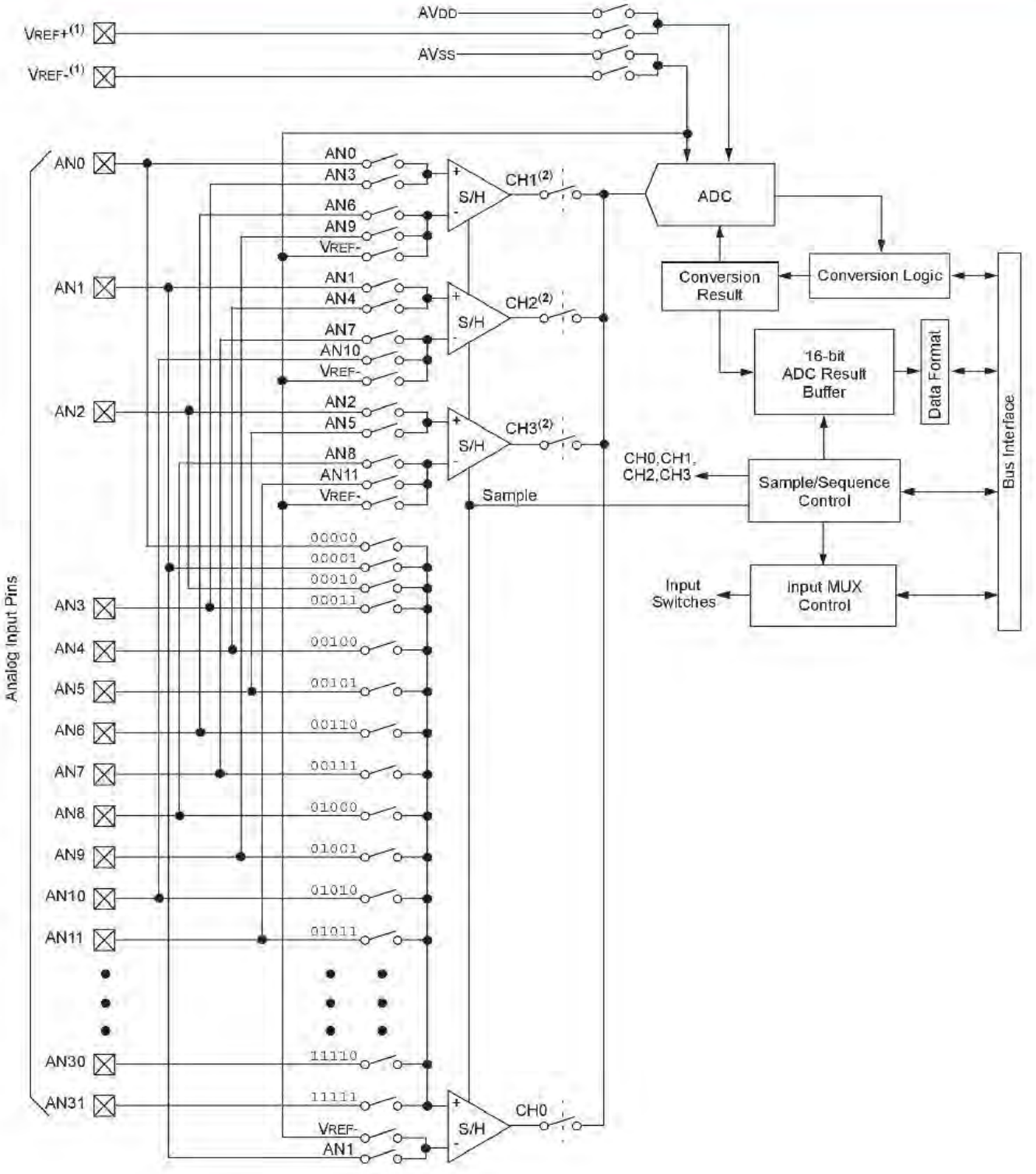



Рис. 1.1. Структурная схема модуля АЦП микроконтроллера

Процесс преобразования аналогового сигнала в цифровой можно представить в виде последовательности двух этапов: этапа выборки и этапа собственно преобразования (рис. 1.2).

Преобразование аналогового сигнала в цифровой код начинается с момента подключения одного из входов АЦП к источнику сигнала (1). На программном уровне это подключение выполняется путём установки бита SAMP регистра управления AD1CON1. Устройство выборки-хранения представляет собой матрицу конденсаторов, которые заряжаются напряжением входного сигнала в течение определённого интервала времени (время выборки). Интервал выборки должен быть достаточно продолжительным, чтобы конденсаторы смогли зарядиться до напряжения, соответствующего входному сигналу. Интервал выборки задаётся программным способом в регистре управления AD1CON3.

По завершении этапа выборки источник входного аналогового сигнала отключается от устройства выборки-хранения, и начинается процесс преобразования накопленного на матрице конденсаторов заряда в цифровой код (2). Для данного этапа предусмотрен целый ряд режимов, позволяющих управлять процессом преобразования вручную или автоматически. Управление этапом преобразования выполняется путём программирования отдельных битов регистра AD1CON1.

На этапе преобразования каждый бит результата выдается по отдельному синхронизирующему импульсу. Для 10 битного преобразования требуется 10 импульсов синхронизации плюс два дополнительных импульса, т. е. всего 12 импульсов. Период таких импульсов (обозначается как TAD) выбирается программно путем установки соответствующих битов в регистре AD1CON3. По завершении преобразования 10 битный цифровой код результата помещается в один из 16 битных буферных регистров, которые доступны для чтения (3). Об окончании преобразования свидетельствует либо установленный бит DONE в регистре управления состояния AD1CON1, либо установленный флаг прерывания АЦП.

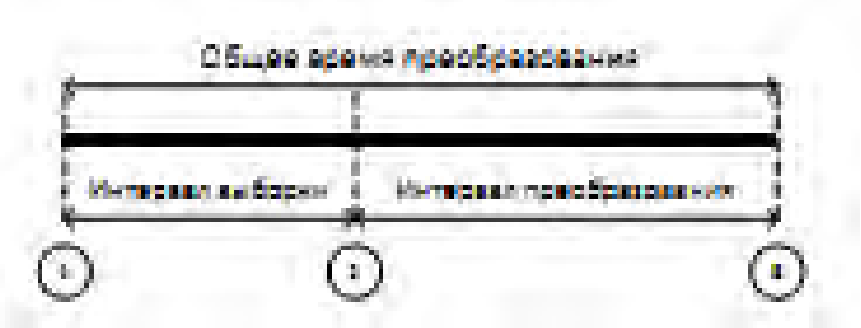


Рис. 1.2. Последовательность выполнения аналого-цифрового преобразования

### 1.3 Программная модель АЦП

С программной точки зрения модуль АЦП представляет собой группу регистров, с помощью которых можно задавать как общие параметры преобразования (интервал выборки, частоту преобразования), так и режимы выполнения самого преобразования (возможность сканирования каналов входных сигналов, автоматического запуска преобразования и т.д.).

Модуль АЦП микроконтроллера dsPIC33fj32mc204 имеет 10 регистров управления состояния:

AD1CON1, AD1CON2, AD1CON3, AD1CON4 – регистры управления;
AD1CHS123 – регистр выбора входа каналов 1, 2, 3;
AD1CHS0 – регистр выбора входа канала 0;
AD1CSSH, AD0CSSL – старший и младший регистры задания последовательности сканирования;
AD1PCFGH, AD0PCFGL – старший и младший регистры конфигурации порта;

Регистры AD1CON1, AD1CON2, AD1CON3 производят настройку модуля АЦП. С их помощью устанавливается частота преобразования, выбирается конфигурация источников опорного напряжения, а также выполняется программное управление выборкой и преобразованием входного сигнала. Регистр AD1CON4 устанавливает количество результатов преобразований для каждого аналогового входа, автоматически сохраняемых в буфер в память микроконтроллера. Регистры AD1CHS123 и AD1CHS0 позволяет выбрать входные каналы, которые будут подключены к устройству выборки-хранения сигнала преобразователя, а также настроить мультиплексор для работы с входными сигналами. Регистры AD1PCFGH и AD1PCFGL производят настройку линий порта микроконтроллера как аналоговые входы либо как цифровые входы/выходы. Регистры AD1CSSH, AD0CSSL позволяют выбрать каналы входных сигналов для режима сканирования.

Биты регистров и назначения некоторых наиболее часто используемых битов указаны в таблицах 1.1 – 1.10.

Таблица 1.1

Биты регистра AD1CON1

| *Обозначение* | ADON | - | ADSIDL | ADDMABM | - | AD12B | FORM<1:0> | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| *Обозначение* | SSRC<2:0> | | | - | SIMSAM | ASAM | SAMP | DONE |
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.2

Назначение некоторых битов регистра AD1CON1

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15 | ADON | Управление работой модуля АЦП<br>ADON = 1: модуль АЦП включен<br>ADON = 0: модуль АЦП отключен |
| 10 | AD12B | Выбор режима работы<br>AD12B = 1: 12-битный режим<br>AD12B = 0: 10-битный режим |
| 9-8 | FORM<1:0> | Задание формата данных результата<br>FORM<1:0> = 00: беззнаковый целочисленный формат |
| 7-5 | SSRC<7:5> | Задание метода инициализации преобразования<br>SSRC<7:5> = 111: Процесс выборки контролируется внутренним счётчиком. По окончании выборки начинается преобразование (автопреобразование).<br>SSRC<7:5> = 010: Окончание выборки контролируется таймером Т3. По окончании выборки начинается преобразование.<br>SSRC<7:5> = 000: Очистка бита выборки SAMP завершает выборку и запускает преобразование |
| 2 | ASAM | Задание автоматического запуска выборки<br>ASAM = 1: Выборка начинается сразу после предыдущего преобразования. Бит SAMP устанавливается автоматически. |
| | | ASAM = 0: Выборка запускается вручную при установке бита SAMP. |

Окончание таблицы 1.2

| | | |
|---|---|---|
| 1 | SAMP | Бит запуска выборки<br>SAMP = 1: Запуск выборки в промежуточные усилители<br>SAMP = 0: Выборка остановлена |
| 0 | DONE | Бит статуса преобразования<br>DONE = 1: Преобразование завершено<br>DONE = 0: Преобразование не запущено либо не завершено |

Таблица 1.3

Биты регистра AD1CON2

| *Обозначение* | VCFG<2:0> | | | - | - | CSCNA | CHPS<1:0> | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | BUFS | - | SMPI<3:0> | | | | BUFM | ALTS |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.4

Назначение некоторых битов регистра AD1CON2

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15-13 | VCFG<2:0> | Выбор источника опорного напряжения<br>VCFG<2:0> = 000 — VREFH: AVDD, VREFL: AVSS<br>VCFG<2:0> = 011 — VREFH: VREF+, VREFL: VREF- |
| 9-8 | CHPS<1:0> | Выбор каналов преобразования<br>CHPS<1:0> = 1x: Преобразование сигналов CH0, CH1, CH2, CH3<br>CHPS<1:0> = 01: Преобразование сигналов CH0, CH1<br>CHPS<1:0> = 00: Преобразование сигнала CH0 |

Таблица 1.5

Биты регистра AD1CON3

| *Обозначение* | ADRC | - | - | SAMC<4:0> | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | ADCS<7:0> | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.6

Назначение некоторых битов регистра AD1CON3

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15 | ADRC | Выбор источника тактирования преобразования<br>ADRC = 0: Внутренняя RC-цепочка<br>ADRC = 1: Системная тактовая частота |

Таблица 1.7

Биты регистра AD1CHS123

| *Обозначение* | - | - | - | - | - | CH123NB<1:0> | | CHS123B |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | - | - | - | - | - | CH123NA<1:0> | | CH123SA |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.8

Назначение некоторых битов регистра AD1CHS123

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 10-9 | CH123NB<1:0> | Выбор отрицательного входа для усилителей CH1, CH2, CH3 для<br>варианта подключения MUXB<br>CH123NB<1:0> = 11: Для CH1 отрицательный вход AN9, для CH2 –<br>AN10, для CH3 –<br>AN11.<br>CH123NB<1:0> = 10: Для CH1 отрицательный вход AN6, для CH2 –<br>AN7, для CH3 – AN8.<br>CH123NB<1:0> = 0x: Для CH1, CH2, CH3 отрицательный вход $V_{REFL}$. |
| 8 | CH123SB | Выбор положительного входа для усилителей CH1, CH2, CH3 для<br>варианта подключения MUXB<br>CH123SB = 1: Для CH1 положительный вход AN3, для CH2 – AN4,<br>для CH3 –<br>AN5.<br>CH123SB = 0: Для CH1 положительный вход AN0, для CH2 – AN1,<br>для CH3 –<br>AN2. |
| 2-1 | CH123NA<1:0> | Выбор отрицательного входа для усилителей CH1, CH2, CH3 для<br>варианта подключения<br>MUXA<br>CH123NA<1:0> = 11: Для CH1 отрицательный вход AN9, для CH2 –<br>AN10, для CH3 –<br>AN11.<br>CH123NA<1:0> = 10: Для CH1 отрицательный вход AN6, для CH2 –<br>AN7, для CH3 – AN8.<br>CH123NA<1:0> = 0x: Для CH1, CH2, CH3 отрицательный вход $V_{REFL}$. |

Окончание таблицы 1.8

| | | |
|---|---|---|
| 8 | CH123SA | Выбор положительного входа для усилителей CH1, CH2, CH3 для<br>варианта подключения<br>MUXA<br>CH123SA = 1: Для CH1 положительный вход AN3, для CH2 – AN4,<br>для CH3 – AN<br>CH123SA = 0: Для CH1 положительный вход AN0, для CH2 – AN1,<br>для CH3 – AN |

Таблица 1.9

Биты регистра AD1CHS0

| *Обозначение* | CH0NB | - | - | CH0SB<4:0> | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | CH0NA | - | - | CH0SA<4:0> | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.10

Назначение некоторых битов регистра AD1CHS0

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15 | CH0NB | Выбор отрицательного входа для усилителя CH0 для выборки B<br>CH0NB = 1: Для CH0 отрицательный вход AN1.<br>CH0NB = 0: Для CH0 отрицательный вход $V_{REFL}$. |
| 12-8 | CH0SB<4:0> | Выбор положительного входа для усилителя CH0 для выборки B<br>CH0SB<4:0> = 01001: Для CH0 положительный вход AN9.<br>…<br>CH0SB<4:0> = 00001: Для CH0 положительный вход AN1.<br>CH0SB<4:0> = 00000: Для CH0 положительный вход AN0. |

Окончание таблицы 1.10

| | | |
|---|---|---|
| 7 | CH0NA | Выбор отрицательного входа для усилителя CH0 для выборки A<br>CH0NA = 1: Для CH0 отрицательный вход AN1.<br>CH0NA = 0: Для CH0 отрицательный вход $V_{REFL}$. |
| 4-0 | CH0SA<4:0> | Выбор положительного входа для усилителя CH0 для выборки A<br>CH0SA<4:0> = 01001: Для CH0 положительный вход AN9.<br>…<br>CH0SA<4:0> = 00001: Для CH0 положительный вход AN1.<br>CH0SA<4:0> = 00000: Для CH0 положительный вход AN0. |

### 1.4 Настройка АЦП

Для настройки модуля АЦП необходимо выполнить следующие действия:

1. Выбрать 10- либо 12-битный режим (ADxCON1<10>).
2. Выбрать источник опорного напряжения (ADxCON2<15:13>).
3. Выбрать источник тактирования модуля АЦП (ADxCON1<7:5>).
4. Установить требуемые линий порта микроконтроллера как аналоговые входы (ADxPCFGH<15:0> и ADxPCFGL<15:0>) .
5. Задать, каким образом аналоговые входы будут подключены к входам усилителей (ADxCHS123<15:0>).
6. Задать число усилителей, которые будут использованы для одновременной выборки (ADxCON2<9:8>, ADxPCFGH<15:0> и ADxPCFGL<15:0>).
7. Задать события, по которому будет происходить выборка каналов (ADxCON1<3>, ADxCSSH<15:0> и ADxCSSL<15:0>).
8. Установить автоматическую либо ручную выборку каналов.
9. Задать события, по которому будет происходить запуск преобразования.
10. Задать каким образом результат преобразования сохраняется в буфер (ADxCON1<9:8>).
11. Задать прерывание или шаг увеличения указателя на буфер DMA сохранения результата преобразования (ADxCON2<9:5>).
12. Задать число выборок в буфере DMA для каждого входа модуля АЦП (ADxCON4<2:0>).
13. Задать формат сохранения результата преобразования.
14. Настроить прерывание модуля АЦП (если требуется).
15. Настроить буферы DMA (если требуется).
16. Включить модуль АЦП (ADxCON1<15>).

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая схема подключения аналоговых сигналов к микроконтроллеру.

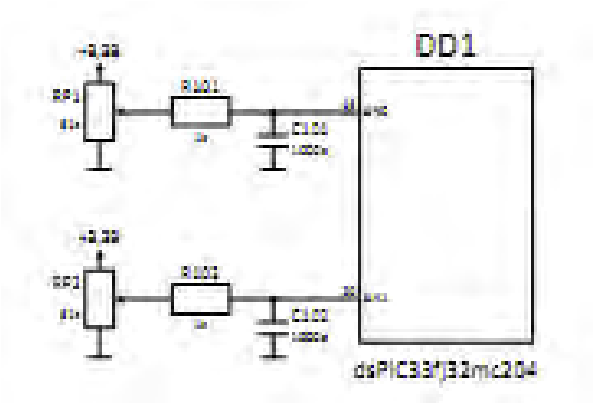



Рис. 2.1. Электрическая схема подключения источников аналогового сигнала

В стенде в качестве источников аналоговых сигналов используются переменные резисторы RP1, RP2, которые через фильтрующие RC-цепочки подключены к входам AN0, AN1 микроконтроллера. Таким образом, напряжение на входе микроконтроллера может изменяться в пределах от 0В до +3.3В.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, отображающую на LED данные аналоговых сигналов с переменных резисторов RP1, RP2.

*Анализ задачи:* Источники аналогового сигнала подключены к входам AN0, AN1. Положительное опорное напряжение – AVdd (+3.3В), отрицательное – AVss (0В). Необходимо вход AN0 подключить к положительному входу усилителя CH0, вход AN1 – к положительному входу усилителя CH1. Отрицательные входы усилителей по умолчанию подключены к отрицательному опорному напряжению, т.е. к AVss. Далее необходимо настроить модуль АЦП на одновременное преобразование сигналов с усилителей CH0, CH1. В основном цикле программы следует периодически запускать процесс аналого-цифрового преобразования, и после его завершения, вывести значения сигналов на LED.

Листинг программы:

```
#include <P33FJ32MC204.h>
#define FOSC
7370000 #define
```

FCY (FOSC / 2)

```
_FOSCSEL(FNOSC_FRC)                 // настройка работы микроконтроллера
                                    // от внутреннего тактового генератора

// объявление кодов сегментов
#define SEG_A       0x08            //       --- SEG_A ---
#define SEG_B       0x01            //       |         |
#define SEG_C       0x20            //    SEG_F     SEG_B
#define SEG_D       0x04            //       |         |
#define SEG_E       0x80            //       --- SEG_G ---
#define SEG_F       0x10            //       |         |
#define SEG_G       0x40            //    SEG_E     SEG_C
#define SEG_DP      0x02            //       |         |

                                    //       --- SEG_D ---   SEG_DP
// объявление кодов цифр
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N0          SEG_F
#define N1          SEG_B + SEG_C
#define N2          SEG_A + SEG_B + SEG_G + SEG_E + SEG_D
#define N3          SEG_A + SEG_B + SEG_G + SEG_C + SEG_D
#define N4          SEG_F + SEG_G + SEG_B + SEG_C
#define N5          SEG_A + SEG_F + SEG_G + SEG_C + SEG_D
                    SEG_A + SEG_F + SEG_G + SEG_C + SEG_D +
#define N6          SEG_E
                    SEG_A + SEG_B +
#define N7          SEG_C
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N8          SEG_F + SEG_G
                    SEG_A + SEG_B + SEG_C + SEG_D + SEG_F +
#define N9          SEG_G

#define DIP         SEG_DP
#define
MINUS               SEG_G

#define OFF         0x00


// массив кодов цифр

char DIGITS[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};
```

```
// Инициализация таймера
T1
    void Init_Timer1(
                void )
{
      T1CON = 0;              // сброс таймера
      IFS0bits.T1IF = 0;      // сброс флага прерывания таймера
                              // разрешение прерывания от
      IEC0bits.T1IE = 1;      таймера
      TMR1                    // обнуление текущего значения
      =       0x0000;         таймера
              0x0E65
      PR1 =         ;         // задание периода таймера
                              // разрешение работы таймера и его
      T1CONbits.TON = 1;      запуск
}


char _ind = 0;      // номер активного индикатора

char _data[8];      // буфер индикации - массив кодов символов

// Отправление данных в регистр
MBI5168 void Ind_Send(char digit)
{
```



```
char c;
          for (c = 0; c < 8; c++)
{
```

- установка требуемого логического уровня

- на последовательном входе SDI драйвера

```
MBI5168 if ((digit & (1 << c)) != 0)
{
LATCbits.LATC3 = 1;

}
else
{

LATCbits.LATC3 = 0;
}
```

- формирование синхроимпульса на входе SCK

```
LATCbits.LATC4 = 1;
LATCbits.LATC4 = 0;

        }

        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
        LATBbits.LATB5 = 0;

    }


    // Инициализация линий портов
    void Ind_Init()
    {
        TRISBbits.TRISB6 = 0;   // Выход A (RB6)
        TRISBbits.TRISB7 = 0;   // Выход B (RB7)
        TRISBbits.TRISB8 = 0;   // Выход C (RB8)
        TRISBbits.TRISB9 = 0;   // Выход D (RB9)
        TRISCbits.TRISC3 = 0;   // Выход SDI (RC3)
        TRISCbits.TRISC4 = 0;   // Выход SCK (RC4)
        TRISBbits.TRISB5 = 0;   // Выход LE (RB5)
    }

    // Разбитие числа на цифры
    void Ind_Show(unsigned int a, unsigned int b)
    {
        _data[3] = DIGITS[a %
        10];                    a /= 10;
        _data[2] = DIGITS[a %
        10];                    a /= 10;
        _data[1] = DIGITS[a %
        10];                    a /= 10;
        _data[0] = DIGITS[a %
        10];
        _data[7] = DIGITS[b %
        10];                    b /= 10;
```

```
        _data[6] = DIGITS[b %
        10];                            b /= 10;
        _data[5] = DIGITS[b %
        10];                            b /= 10;
        _data[4] = DIGITS[b %
        10];
}
// Прерывание таймера T1 по совпадению
void __attribute__((interrupt,no_auto_psv)) _T1Interrupt( void )
{
        Ind_Send(OFF);                  // Отключение индикатора
        LATB &= ~(0x07 <<
        6);                             // Переключение на следующий
        LATB |= (_ind <<
        6);                             // индикатор

        Ind_Send(_data[_ind]);          // Отправка кода цифры

        _ind++;                         // переход к след. индикатору
        if (_ind == 8)
        {
                _ind = 0;
        }

        IFS0bits.T1IF = 0;      // Сброс флага прерывания таймера
        TMR1      = 0;          // Перезапуск таймера
}

void
ADC_Init()
{
        AD1CON1bits.AD12B
        = 0;                    // 10-битный режим
        AD1CON2bits.V
        CFG             = 0;    // Опорное напряжение – Avdd
        AD1CON1bits.SS          // Запуск преобразования после
        RC              = 0;    очистки
                                // бита выборки
        AD1PCF
        GL              = 0xFFFF;
        AD1PCFGLbits.PCFG
        0 = 0;                  // AN0 - аналоговый вход
        AD1PCFGLbits.PCFG
        1 = 0;                  // AN1 - аналоговый вход
        AD1CON1bits.AS          // Автоматический запуск выборки
        AM              = 1;    после
```

```
                                    // предыдущего преобразования
        AD1CON3bits.A               // Источник тактирования
        DRC             = 0;        преобразования -
                                    // источник тактирования МК
        AD1CON1bits.FO              // Формат данных - целый
        RM              = 0;        беззнаковый
        AD1CON2bits.C
        HPS             = 1;        // Преобразование каналов CH0, CH1
        AD1CON1bits.SIMSA
        M= 1;                       // Одновременная выборка CH0, CH1

        AD1CHS123bits.CH123SA = 0;// Подключение входа AN0 на
                                    положительный // вход усилителя CH0

        AD1CHS0bits.CH0SA = 1; // Подключение входа AN1 на
                                    положительный // вход усилителя CH1

        AD1CON1bits.ADON
        = 1;                        // Включение модуля АЦП
}

void main()
{
        ADC_Init();                 // Инициализация АЦП


        Ind_Init();                 // Инициализация индикации

        Init_Timer1();              // Инициализация таймера


        while (1)
        {
                AD1CON1bits.SAMP =
                0;                              // Запуск выборки
                while
                (!AD1CON1bits.DONE)             // Ожидание
                ;                               завершения
                                                // преобразования
                Ind_Show(ADC1BUF1,              // Отображение
                ADC1BUF0);                      результата
                __delay32(FCY / 10);
        }

}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Отобразить на LED сумму значений аналоговых сигналов RP1 и RP2.
2. Отобразить на LED мгновенное и усреднённое за 2 секунды значение аналогового сигнала RP1.
3. Отобразить на LCD мгновенное и усреднённое за 5 секунд значение аналогового сигнала RP2.
4. Отобразить на VD2..VD10 значение аналогового сигнала RP1 в двоичном виде. В случае RP1 > RP2, вывести на LCD предупреждение.
5. Отобразить значение аналогового сигнала RP1 на VD2..VD10, используя VD2..VD10 в качестве линейной шкалы. В случае RP1 < RP2, вывести на LED предупреждение.

## 5 Контрольные вопросы

1. Для чего предназначены аналого-цифровые преобразователи?
2. Какие существуют схемы построения АЦП?
3. Какие достоинства и недостатки присущи параллельным АЦП?
4. Какими основными характеристиками обладают АЦП?
5. Какие параметры АЦП микроконтроллера dsPIC33fj32mc204?
6. Каким образом происходит настройка модуля АЦП микроконтроллера?

# Лабораторная работа 13

## Исследование работы энергонезависимой памяти и интерфейса $I^2C$

### Цель работы:

Изучить принцип функционирования интерфейса $I^2C$, а так же алгоритм и принцип работы энергонезависимой памяти. Разработать и отладить программу для обмена информацией между микроконтроллером и микросхемой внешней памяти.

### Порядок выполнения работы:

1. Изучить теоретические вопросы, связанные с принципом функционирования интерфейса $I^2C$ и работой энергонезависимой памяти.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу микросхемы внешней памяти.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Интерфейс $I^2C$

Интерфейс $I^2C$ (Inter-Integrated Circuit) является одним из наиболее популярных среди разработчиков оборудования. В настоящее время выпускается огромное число микросхем, использующих этот интерфейс. Стандарт $I^2C$ реализован как двухпроводной последовательный интерфейс, разработанный компанией «Philips Corp.» для работы с максимальной скоростью передачи данных 100 Кбит/с. Впоследствии стандарт стал поддерживать более скоростные режимы работы шины (400 Кбит/с и 1Мбит/с). При этом к одной шине $I^2C$ могут быть подключены устройства с различными скоростями доступа, если скорость передачи данных будет удовлетворять требованиям самого низкоскоростного устройства.

Протокол передачи данных по шине $I^2C$ разработан таким образом, чтобы гарантировать надежный качественный прием/передачу данных. При обмене данными одно устройство является «ведущим» и инициирует такой обмен, а также формирует сигналы синхронизации. Другое устройство, «ведомое», может начать передачу/прием данных только по команде ведущего шины.

Протокол I²C использует две сигнальные линии, по одной из которых, обычно обозначаемой как SCL, подается сигнал синхронизации, а по другой, обозначаемой обычно как SDA, передаются или принимаются данные. К шине I²C можно подключать несколько устройств, при этом линии SCL и SDA являются общими (рис. 1.1).

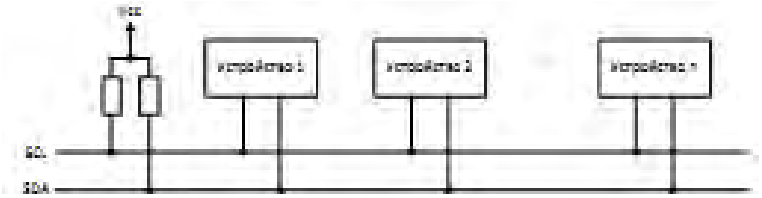


Рис. 1.1. Схема подключения устройств по интерфейсу I²C

К сигнальным линиям необходимо подключить так называемые подтягивающие резисторы, обозначенные на схеме как Rp, присоединив их к источнику питания. Подтягивающие резисторы нужны для фиксации уровня сигналов, поскольку спецификация I²C предусматривает использование устройств, имеющих выходные каскады с открытым коллектором или стоком (open drain). Кроме того, что более существенно, поскольку линии SCL и SDA являются двунаправленными, то такая аппаратная конфигурация обеспечивает функцию «монтажного» И (AND), что позволяет передавать сигналы в обоих направлениях.

Конфигурация шины I²C позволяет работать с одним или несколькими ведущими, что дает возможность создавать разветвленные высокоскоростные сети обмена данными. В подавляющем большинстве случаев на шине находится только одно ведущее и несколько ведомых устройств. «Ведущий» обеспечивает синхронизацию обмена данными, генерируя синхроимпульсы по линии SCL.

Обмен данными по шине I²C инициируется ведущим устройством, которое должно обеспечить формирование стартовой (в начале обмена) и стоповой (в конце обмена) последовательностей сигналов (рис. 1.2).

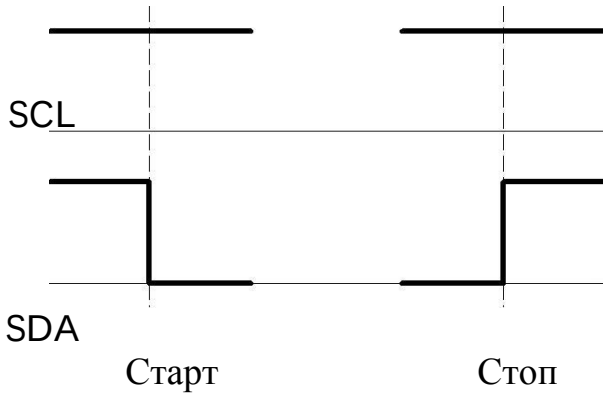



Рис. 1.2. Стартовая и стоповая последовательности сигналов интерфейса I²C

Протокол $I^2C$ требует, чтобы сигнал на линии данных SDA оставался неизменным при ВЫСОКОМ уровне сигнала SCL. Перепад сигнала на линии SDA из ВЫСОКОГО уровня в НИЗКИЙ при ВЫСОКОМ уровне на линии SCL формируется «ведущим» и указывает на начало обмена данными. Если при ВЫСОКОМ уровне сигнала на линии SCL уровень сигнала на линии SDA меняется с НИЗКОГО на ВЫСОКИЙ, то эта последовательность сигналов интерпретируется как завершение операции обмена данными на шине.

Данные передаются по шине в виде последовательности из 8 бит, причем первым в последовательности идет старший значащий бит (MSB). В практическом плане каждый бит удобно передавать по нарастающему фронту сигнала SCL. После того как все 8 бит переданы, устройство, передающее данные, ожидает от устройства, принимающего данные, подтверждения приема (acknowledge - ACK) по линии SDA. Бит ответа фиксируется по нарастающему фронту 9-го импульса SCL. В момент передачи бита подтверждения отвечающее устройство принимает управление линией SDA на себя, а затем возвращает управление инициатору обмена.

Поскольку к шине $I^2C$ можно подключать несколько устройств, то для обмена данными с конкретным устройством вначале нужно указать его адрес. В простейшем случае, например, для передачи данных в устройство вначале нужно передать адрес, а затем и сам байт данных. Цикл передачи данных в устройство в таком случае начинается со стартовой последовательности. Затем передаются первые 8 бит, содержащие 7-разрядный адрес устройства (биты A7…A1) и бит команды чтения-записи (обозначается как R/W). Обычно команда записи передается низким уровнем, а команда чтения — высоким, но для микросхем различных производителей могут быть отличия. Бит 9 — это ответ устройства (ACK), который фиксируется по фронту 9-го синхроимпульса и анализируется «ведущим». Следующие 8 бит являются битами данных. Как и при передаче адреса, 9-й бит также содержит ответ устройства. Цикл записи оканчивается стоповой последовательностью.

### 1.2 Подтверждение приёма

Как уже было упомянуто, передача 8 бит данных от передатчика к приемнику завершаются дополнительным циклом (формированием 9-го тактового импульса линии SCL), при котором приемник выставляет низкий уровень сигнала на линии SDA, как признак успешного приема байта.

Подтверждение при передаче данных обязательно, кроме случаев окончания передачи ведомой стороной. Соответствующий импульс синхронизации генерируется ведущим. Передатчик отпускает линию SDA на время синхроимпульса подтверждения. Приёмник должен удерживать линию SDA в стабильном НИЗКОМ состоянии в течение ВЫСОКОГО состояния синхроимпульса подтверждения.

### 1.3 Адресация в шине I²C

Каждое устройство на шине I²C имеет собственный идентификатор – адрес устройства. Часто производителями микросхем 7-битный адрес делится на две части: контрольный код и биты выбора микросхем. Контрольный код одинаков для однотипных микросхем, а биты выбора микросхемы реализованы в виде дополнительных цифровых входов селектора адреса. Способ подключения дополнительных цифровых входов и определяет биты выбора микросхемы. Таким образом, имеется возможность подключения к одной линии данных нескольких однотипных устройств.

### 1.4 Микросхема энергонезависимой памяти AT24C128

Микросхема AT24C128 представляет собой электрически стираемое и перепрограммируемое постоянное запоминающее устройство (ППЗУ), содержащее 16384 8-битных ячеек. Память организована в виде 256 страниц по 64 байта. Износостойкость ячеек памяти – не менее 1 миллиона циклов записи. Цикл записи данных длится не более 5 мс. Условное обозначение микросхемы приведено на рис. 1.3, назначение выводов представлено в таблице 1.1.

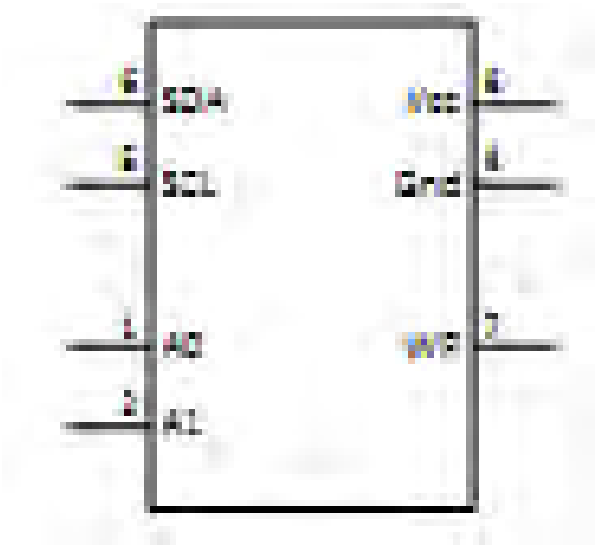



Рис. 1.3. Условное обозначение микросхемы AT24C128

Таблица 1.1

Назначение выводов микросхемы AT24C128

| *Номер вывода* | *Обозначение* | *Назначение* |
|---|---|---|
| 8 | Vcc | Напряжение питания |
| 4 | Gnd | Общий провод |
| 5, 6 | SCL, SDA | Линии интерфейса I²C |
| 1, 2 | A0, A1 | Дополнительные адресные входы |
| 7 | WP | Вход защиты от записи |

Управление микросхемой осуществляется по интерфейсу $I^2C$. Микросхема имеет 7-битный идентификатор, старшие 4 бита которого равны 10100. Младшие 2 бита определяются состоянием входов A1, A0.

Любое обращение к микросхеме по шине $I^2C$ начинается с цикла посылки идентификатора устройства.

**Операция записи байта**

Для выполнения операции записи требуется дополнительно передать два 8-разрядных слова адреса. После приема адреса микросхема памяти отвечает нулем, затем при поступлении импульсов синхронизации принимает 8 бит слова данных. После приема 8 бит слова данных микросхема передает подтверждение приёма. После этого ведущее устройство прерывает последовательность записи передачей условия стоп. В этот же момент времени в EEPROM активизируется внутренне синхронизируемый цикл записи в энергонезависимую память. Все входы отключаются в процессе выполнения цикла записи, и до завершения записи микросхема не реагирует на внешние запросы (рис. 1.4).

Рис. 1.4. Запись байта

**Страничная запись**

Микросхема памяти AT24C128 поддерживают запись 64-байтных страниц. Страничная запись инициируется таким же способом, что и побайтная запись, за исключением того, что ведомое устройство не отправляет условие останова после приема первого слова данных. Взамен этого, микросхема памяти подтверждает прием первого слова данных, после чего может передавать до 63 слов данных. После приема каждого последующего слова данных микросхема памяти передает подтверждение приёма. Ведомое устройство прекращает последовательность страничной записи путем передачи условия останова.

**Опрос подтверждения**

Сразу после инициирования внутренне-синхронизируемого цикла записи и отключения входов микросхемы памяти можно начать опрос подтверждения. Для этого необходимо отправить условие старта после адресного слова. Бит чтения/записи определяет выполнение желаемой

операции. Микросхема памяти передает подтверждение приёма только по завершении внутреннего цикла записи, тем самым, позволяя продолжить последовательность записи.

Операции чтения инициируются тем же способом, что операции записи за исключением того, что бит выбора операции чтения/записи в адресном слове равен единице. Поддерживаются три операции чтения: чтение по текущему адресу, чтение по произвольному адресу и упорядоченное чтение.

**Чтение по текущему адресу**

Внутренний счетчик адреса слова данных хранит адрес, который использовался при последней операции чтения или записи, увеличенный на 1. Данный адрес остается действительным то тех пор, пока на микросхему подано питание. При выполнении инкрементирования после чтения последнего байта последней страницы, устанавливается адрес первого байта первой страницы.

Сразу после приема адреса микросхемы с битом выбора чтения/записи равным единице и подтверждения приема со стороны микросхемы памяти передается слово данных по текущему адресу. Ведущее устройство не отвечает подтверждением, а передает условие останова (рис. 1.5).

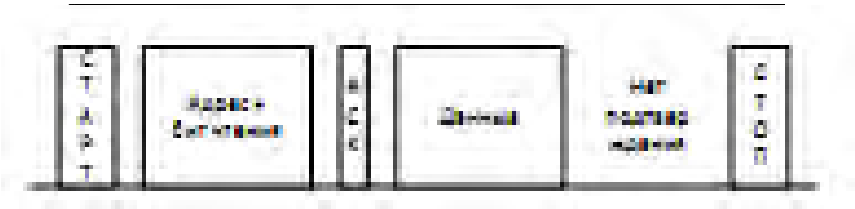


Рис. 1.5. Чтение по текущему адресу

**Чтение по произвольному адресу**

Для чтения по произвольному адресу необходимо предварительно выполнить процедуру загрузки адресного слова данных. Как только микросхема примет адресное слово микросхемы, адресное слово данных и отправит подтверждение, микроконтроллер должен генерировать новое условие старта. После этого необходимо инициировать операцию чтения по текущему адресу путем отправки адреса микросхемы с установленным в единичное состояние битом выбора чтения/записи. Микросхема памяти подтверждает адрес микросхемы и последовательно передает слово данных. Ведомое устройство не отвечает нулем, а должно генерировать условие останова (рис 1.6).

Рис. 1.6. Чтение по произвольному адресу

**Упорядоченное чтение**

Упорядоченное чтение инициируется после операции чтения по текущему адресу или после операции чтения по произвольному адресу. После приема ведущим устройством слова данных оно отвечает подтверждением. Когда микросхема памяти примет подтверждение, выполняется инкрементирование адреса слова данных и передается очередное слово данных. По достижении границы адресного пространства адрес слова данных переходит в начальное состояние и упорядоченное чтение продолжится. Упорядоченное чтение прекращается, когда микроконтроллер не отвечает подтверждением, но продолжает генерировать условия останова (рис. 1.7).

Рис. 1.7. Упорядоченное чтение

**2 Электрическая принципиальная схема к лабораторной работе**

На рис.2.1 приведена электрическая схема подключения микросхемы энергонезависимой памяти к микроконтроллеру по интерфейсу $I^2C$.

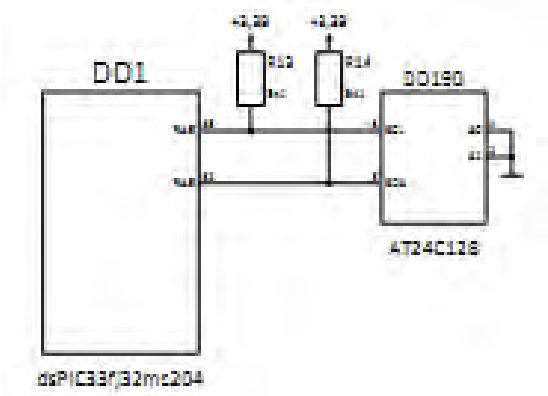



Рис. 2.1. Электрическая схема подключения микросхемы памяти

Линия тактирования SCL подключена к порту RA9 микроконтроллера, линия данных SDA – к порту RA8. Линии SCL и SDA имеют подтягивающий резистор.

Адрес микросхемы формируется из фиксированной части и битов, формируемых аппаратно в зависимости от схемы подключения дополнительных адресных входов микросхемы. Фиксированная часть представляет собой 10100 в двоичном коде. Дополнительные адресные входы имеют нулевой потенциал, соответственно вторая часть адреса равна 00. Таким образом, при инициализации цикла чтения управляющее слово будет 10100000 (A0 в шестнадцатеричном коде), а при инициализации цикла записи – 10100001 (A1 в шестнадцатеричном коде).

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую с клавиатуры вводить последовательно адрес ячейки и число для записи в данную ячейку памяти. Клавишу «#» использовать как подтверждение ввода, клавишу «*» – для отмены ввода. После записи данных в ПЗУ ввести адрес для чтения, произвести чтение данных из ячейки памяти по введённому адресу, и вывести результат. Для отображения информации использовать ЖКИ.

*Анализ задачи:* Операции по работе с интерфейсом $I^2C$ (чтение, запись данных, формирование стартовой и стопой последовательности) удобней организовать в виде соответствующих функций. Для определения нажатой клавиши используется двумерный массив, каждый элемент которого содержит код кнопки, расположенной на пересечении соответствующей строки и столбца матричной клавиатуры.

Листинг программы:

```
#include
<P33FJ32MC204.h>
#include <stdlib.h>
#define FOSC
7370000 #define
FCY (FOSC / 2)
#define Delay_ms(d) (__delay32 (((d)) * ((FCY) /
1000uL))) #define Delay_us(d) (__delay32 (((d)) *
((FCY) / 1000000uL)))
_FOSCSEL(FNOSC_FR
C)                              // настройка работы микроконтроллера
                                // от внутреннего тактового генератора
#define
LCD_EN          LATAbits.LATA10
#define
LCD_RW          LATCbits.LATC2
#define
LCD_RS          LATAbits.LATA7
```

```
#define
LCD_DB7         LATBbits.LATB12
#define
LCD_DB6         LATBbits.LATB14
#define
LCD_DB5         LATBbits.LATB11
#define
LCD_DB4         LATBbits.LATB10


// Подпрограмма отправки тетрады данных в
драйвер ЖКИ void LCD_set_half_byte(char x)

{

        if (x & (0x01 <<
                3))
                LCD_DB7
                = 1;

        else
                LCD_DB7
                = 0;
        if (x & (0x01 <<
        2))
                LCD_DB6
                = 1;
        else
                LCD_DB6
                = 0;
        if (x & (0x01 <<
        1))
                LCD_DB5
                = 1;
        else
                LCD_DB5
                = 0;
        if (x & (0x01 <<
        0))
                LCD_DB4
                = 1;
        else
                LCD_DB4
                = 0;
```

```
                        Delay_ms(
        LCD_EN = 1;     1);
                        Delay_ms(
        LCD_EN = 0;     1);
}
// Инициализация используемых линий
порта
void LCD_InitPorts()
{
        TRISBbits.TRISB                         O
        1               = 0;    // Out          E   (RB1)
        LATBbits.LATB1  = 0;    // OE           Lo
        TRISAbits.TRISA
        10              = 0;    // Out          En (RA10)
        TRISCbits.TRISC
        2               = 0;    // Out          R/W (RC2)
        TRISAbits.TRISA                         R
        7               = 0;    // Out          S   (RA7)
        TRISBbits.TRISB
        12              = 0;    // Out          DB7 (RB12)
        TRISBbits.TRISB
        14              = 0;    // Out          DB6 (RB14)
        TRISBbits.TRISB
        11              = 0;    // Out          DB5 (RB11)
        TRISBbits.TRISB
        10              = 0;    // Out          DB4   (RB10)
}
// Инициализация контроллера
ЖКИ
void LCD_Init()
{
        LCD_RS = 0;
        LCD_RW = 0;
        LCD_EN = 0;
        Delay_ms(30);

        LCD_set_half_byte(0x03);
        Delay_ms(5);
        LCD_set_half_byte(0x03);
        Delay_ms(100);

        LCD_set_half_byte(0x03);

        LCD_set_half_byte(0x02);
```

```
        // 4-битный интерфейс, 2 строки, размер символа - 5x8

        LCD_set_half_byte(0
        x02);
        LCD_set_half_byte(0
        x08); Delay_ms(37);

        // Курсор выкл., мигание позиции выкл., индикатор вкл.

        LCD_set_half_byte(0
        x00);
        LCD_set_half_byte(0
        x0C); Delay_ms(37);

        // Очитска дисплея

        LCD_set_half_byte(0x00);
        LCD_set_half_byte(0x01);
        Delay_ms(2);

        // Направление перемещения курсора - вправо

        LCD_set_half_byte(0x00);
        LCD_set_half_byte(0x06);
}
```

4.1 Отправление данных в DDRAM контроллера ЖКИ void LCD_send_char(char c)

```
{

        LCD_RS = 1;
        LCD_set_half_byte(c >>
        4); LCD_set_half_byte(c
        & 0x0f); LCD_RS = 0;

}
```

4.2 Вывод строки на ЖКИ

```
void LCD_WriteString(char str[])
{
        char i = 0;
```

```
        while (str[i] != 0)

        {

                LCD_send_char(str
                [i]); i++;

        }
}

// Вывод числа на ЖКИ
void LCD_WriteValue(unsigned int value)

{

        char strtmp[16];
        utoa(strtmp, value,
        10);
        LCD_WriteString(strt
        mp);
}
```

5.1 Перевод курсора ЖКИ в начальную позицию void LCD_Home()

```
{

        LCD_RS = 0;
        LCD_set_half_byte(0x02
        >> 4);
        LCD_set_half_byte(0x02 &
        0x0f);
}
```

5.2 Очистка ЖКИ

```
void LCD_Clear()

{

        LCD_RS = 0;
        LCD_set_half_byte(0x01
        >> 4);
        LCD_set_half_byte(0x01 &
        0x0f);
```

```
}
                        TRISAbits.TRIS
#define SDA                     A8
                        TRISAbits.TRIS
#define SCL                     A9
#define READ_SDA   PORTAbits.RA8

// Инициализация портов I2C
void I2C_Init()
{
        ODCAbits.ODC                    // Открытый коллектор
        A8                  = 1;        SDA
        ODCAbits.ODC                    // Открытый коллектор
        A9                  = 1;        SCL
        LATAbits.LAT
        A8                  = 0;
        LATAbits.LAT
        A9                  = 0;

        SDA = 1;

        SCL = 1;

}

// Формирование условия
СТАРТ void I2C_Start()

{
        SDA = 0;    Delay_us(25);

}

// Повторное формирование условия
СТАРТ void I2C_ReStart()

{
        SCL = 0;    Delay_us(25);

        SDA = 1;    Delay_us(25);
        SCL = 1;    Delay_us(25);
        SDA = 0;    Delay_us(25);

        SCL = 0;    Delay_us(25);
```

```
}

// Формирование условия
СТОП void I2C_Stop()
{
        SCL = 1;    Delay_us(25);
        SDA = 1;    Delay_us(25);
}
// Передача байта по
интерфейсу I2C char
I2C_Send(unsigned char c)
{
        char b;
        for (b = 7; b >= 0; b--)
        {
                SCL = 0;    Delay_us(25);

                if ((c & (1 << b)) != 0)

                {
                        SDA = 1;
                }

                else
                {
                        SDA = 0;
                }
                Delay_us(25);

                SCL = 1;    Delay_us(50);
        }
        1. ACK
        SCL =
        0;

        2. Отпустить линию
        SDA = 1;    Delay_us(25);
        SCL = 1;    Delay_us(25);

        char result;
        if (READ_SDA == 0)
        {
                result = 0;

        }
```

```
        else
        {
                result = 1;
        }
        SCL = 0;     Delay_us(25);

        SDA = 0;     Delay_us(25);

        return result;
}

// Приѐм байта по
интерфейсу I2C char
I2C_Read(char ack)
{
        SCL = 0;
        SDA = 1;

        Delay_us(50);

        char b, data = 0;
        for (b = 0; b < 8; b++)
        {

                SCL = 1;
                Delay_us(2
                5); data
                <<= 1;

                if (READ_SDA != 0)
                {

                        data |= 1;
                }
                Delay_us(25);

                SCL = 0;     Delay_us(50);
        }

        SDA = 1;
        SCL = 1;

        return data;

}
```

```
// Запись байта в ПЗУ

void EE_WriteByte(unsigned int addr, char data)
{

        char addrhi = (char)(addr >> 8);
        char addrlo = (char)(addr &
        0x00FF);

        I2C_Start();

        I2C_Send(0xA0);
        I2C_Send(addrhi);
        I2C_Send(addrlo);
        I2C_Send(data);
        I2C_Stop();

        Delay_us(10);
}

// Чтение байта из ПЗУ
char EE_ReadByte(unsigned int addr)

{

        char addrhi = (char)(addr >> 8);
        char addrlo = (char)(addr &
        0x00FF);

        I2C_Start();
        I2C_Send(0xA0);

        I2C_Send(addrhi);

        I2C_Send(addrlo);

        I2C_ReStart();
        I2C_Send(0xA1);
        char data =
        I2C_Read(0);
        I2C_Stop();

        return data;
```

```
}
                                        // номер активного вывода
char _i = 0;                            дешифратора
char _keys[4][3] = {{9, 8, 7},          // массив кнопок клавиатуры
                                    {6, 5, 4},
                                    {3, 2, 1},
                                    {'#', 0, '*'}};
char _key = 0;                                  // текущая нажатая кнопка
char _keyPressed = 0;                           // флаг нажатия кнопки

// Инициализация таймера
T1
void Init_Timer1()
{
        T1CO                    // сброс
        N           = 0;        таймера
        IFS0bits.T1IF = 0;      // сброс флага прерывания таймера
        IEC0bits.T1IE = 1;      // разрешение прерывания от таймера
        TMR
        1           = 0x0000;   // обнуление текущего значения таймера
        PR1         = 0x0E65;   // задание периода таймера
        T1CONbits.TON = 1;      // разрешение работы таймера и его запуск
}

// Инициализация линий портов клавиатуры
void Kbd_Init()
{
        TRISBbits.TRISB6 = 0;       // Выход    A     (RB6)
        TRISBbits.TRISB7 = 0;       // Выход    B     (RB7)
        TRISBbits.TRISB8 = 0;       // Выход    C     (RB8)
        TRISBbits.TRISB9 = 0;       // Выход    D     (RB9)


        TRISCbits.TRISC6 = 0;       // Out      INA   (RC6)
        TRISCbits.TRISC7 = 0;       // Out      INB   (RC7)
        TRISCbits.TRISC8 = 0;       // Out      INC   (RC8)

        TRISCbits.TRISC9 = 1;       // In       D     (RC9)
}

void Kbd_ReadCol(char row)
{
        char column;
        for (column = 0; column < 3; column++)
        {
                LATC &= ~(0x07 << 6);               // Выбор входа
                LATC |= (column << 6);
```

```
            __delay32(10L);                 // Пауза для установки сигнала
            // чтение
            столбца
            if
            (!PORTCbits.RC
            9)
            {
                  _key = _keys[row][column]; // Сохранение нажатой
                  кнопки
                  _keyPressed = 1;          // Установка флага нажатия
                  кнопки

            }
      }
}

// Прерывание таймера T1 по совпадению

void __attribute__((interrupt,no_auto_psv)) _T1Interrupt()

{

                                          // Переключение на
      LATB &= ~(0x0F << 6);               следующий
      LATB |= (_i + 8) << 6;              // выход дешифратора

      Kbd_ReadCol(_i);                    // Сканирование столбца

      _i++;
      if (_i == 4)
      {
            _i = 0;
            if (!_keyPressed)
            {
                  _key = -1;
            }
            _keyPressed = 0;              // Сброс флага нажатия кнопки
      }
                                          // Сброс флага прерывания
      IFS0bits.T1IF = 0;                  таймера
      TMR
      1           = 0;                    // Перезапуск таймера

}

// Ввод данных с клавиатуры с отображением
```

на ЖКИ unsigned int InputValue(char *str)
{

```
static char prevKey
= 0; unsigned int
value = 0;

LCD_Clear();
LCD_WriteString(str);

LCD_WriteValue(value);

while (1)

{

        if (prevKey != _key)
        {

                prevKey =
                _key; if (_key
                == '*')
                {

                        value = value / 10;

                }
                else if (_key == '#')

                {
                        return value;
                }
                else if (_key != -1)
                {

                        value = value * 10 + _key;

                }

                LCD_Clear();
                LCD_WriteString(st
                r);
                LCD_WriteValue(v
                alue);
```

```
            }
        }

}


void main()
{
        AD1PCFGL = 0xFFFF;          // Отключение входов АЦП

        Kbd_Init();                 // Инициализация портов
        клавиатуры
        Init_Timer1();              // Инициализация таймера
        I2C_Init();                 // Инициализация портов I2C
        LCD_InitPorts();          // Инициализация портов для управления
        ЖКИ
        LCD_Init();                 // Инициализация ЖКИ

        while (1)
        {

                // Ввод адреса и данных
                unsigned int addr = InputValue("Write addr:\0");
                unsigned char data = (unsigned char)InputValue("Write
                data:\0");

                1.Сохранение данные в ПЗУ

                EE_WriteByte(addr,
                data); LCD_Clear();
                LCD_WriteString("O
                k"); __delay32(FCY);

                2.Ввод адреса чтения

                LCD_Clear();

                addr = InputValue("Read addr:\0");

                // Чтение данных из
                ПЗУ data =
                EE_ReadByte(addr);
                LCD_Clear();

                LCD_WriteString("Read
```

```
                data:\0");
                LCD_WriteValue(data);
                __delay32(2 * FCY);

            }

        }
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнять следующие действия:

1. Записать состояние тумблеров SA3..SA10 в память ПЗУ. Адрес ячейки памяти ввести с клавиатуры.
2. Отобразить значение ячейки памяти ПЗУ с помощью светодиодов VD3..VD10. Адрес ячейки памяти ввести с клавиатуры.
3. Записать значение аналогового сигнала с датчика RP1 в две ячейки памяти ПЗУ. Адрес младшей ячейки памяти ввести с клавиатуры.
4. Отобразить на ЖКИ значение 8 последовательных ячеек памяти ПЗУ. Адрес начальной ячейки памяти ввести с клавиатуры.

## 5 Контрольные вопросы

1. Для чего предназначена шина $I^2C$?
2. Дайте основные характеристики шины $I^2C$.
3. Каково назначение линий шины $I^2C$?
4. Каким образом происходит адресация устройств?
5. Какое количество устройств может быть одновременно подключено к шине $I^2C$?
6. Каким образом происходит передача данных по шине $I^2C$?
7. Каково назначение микросхемы памяти AT24C128?
8. Каким образом происходит чтение данных из микросхемы памяти?
9. Каким образом происходит запись данных в микросхему памяти?

# Лабораторная работа 14

## Исследование средств вывода аналоговой информации

**Цель работы:**

Изучить способы вывода аналоговой информации. Изучить схему сопряжения микросхемы цифро-аналого преобразователя с микроконтроллером.

**Порядок выполнения работы**:

1. Изучить теоретические вопросы, связанные с принципом работы цифро-аналогового преобразователя.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу системы вывода аналогового сигнала.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Цифро-аналоговое преобразование

Цифро-аналоговый преобразователь (ЦАП) — устройство для преобразования цифрового кода в аналоговый сигнал (ток, напряжение либо сопротивление). Цифро-аналоговые преобразователи применяются для сопряжения цифровых устройств с аналоговыми схемами.

Применяются в основном 2 метода цифро-аналогового преобразования: суммирование единичных эталонных величин и суммирование эталонных величин, веса которых различаются. В первом для формирования выходной величины используется только одна эталонная величина, во втором методе применяются эталонные величины с разными весами, зависящими от номера разряда, и суммируются только те эталонные величины, для которых в соответствующем разряде входного кода установлена единица.

Основной характеристикой ЦАП является разрядность. Разрядность — это количество различных уровней выходного сигнала, которые ЦАП может воспроизвести. Разрядность обычно задается в битах; количество бит есть логарифм по основанию 2 от количества уровней. Например, однобитный ЦАП способен воспроизвести два уровня, а восьмибитный — 256 уровней.

### 1.2 Микросхема ЦАП AD5241

Микросхема AD5241 представляет собой переменное сопротивление с 256 уровнями квантования, конфигурируемое по интерфейсу $I^2C$. Условное обозначение микросхемы приведено на рис.1.1, назначение выводов представлено в таблице 1.1.

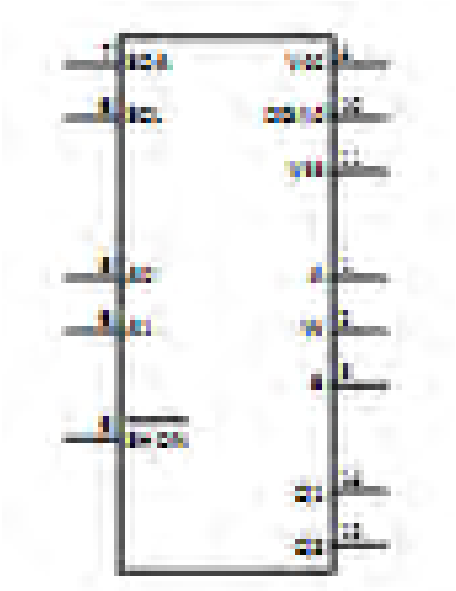


Рис. 1.1. Условное обозначение микросхемы AD5241

Таблица 1.1

Назначение выводов микросхемы AD5241

| *Номер вывода* | *Обозначение* | *Назначение* |
|---|---|---|
| 1, 2, 3 | A, W, B | Выводы потенциометра |
| 4 | Vcc | Напряжение питания |
| 10 | DGnd | Общий провод цифровой части |
| 11 | Vss | Общий провод аналоговой части |
| 6, 7 | SCL, SDA | Линии интерфейса $I^2C$ |
| 8, 9 | A0, A1 | Дополнительные адресные входы |
| 14, 12 | Q1, Q2 | Дополнительные дискретные выходы |

Полное сопротивление между выводами A и B может быть 10кОм, 100кОм, 1Мом (в зависимости от модели микросхемы). Сопротивление разбито на 256 частей, через внутренние ключи подсоединённые к выводу 8-разрядного регистра. Регистр декодирует поступающий цифровой код, коммутируя аналоговый выход микросхемы с выводом соответствующего ключа. Таким образом, микросхему можно представить в виде потенциометра с задаваемым в цифровом виде положением среднего вывода. Напряжение на аналоговом выходе микросхемы при этом можно вычислить по формуле:

Управление микросхемой осуществляется по интерфейсу I²C. Любое обращение к микросхеме начинается с цикла посылки идентификатора устройства. После получения подтверждения от микросхемы необходимо отослать управляющий байт. Формат управляющего байта представлен в таблице 1.1.

Таблица 1.1

Формат управляющего байта

| *Номер бита* | *Обозначение* | *Назначение* |
|---|---|---|
| 7 | A/B | Выбор одного из двух внутренних регистров. |
| 6 | RS | Сброс. Устанавливает аналоговый выход в среднее положение. |
| 5 | SD | Выключение. Аналоговый выход подключается к выводу B. |
| 4 | O1 | Бит управления дискретным выходом Q1. |
| 3 | O2 | Бит управления дискретным выходом Q2. |
| 2 | – | Не используются. |
| 1 | – | |
| 0 | – | |

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая схема подключения микросхемы ЦАП к микроконтроллеру по интерфейсу I²C.

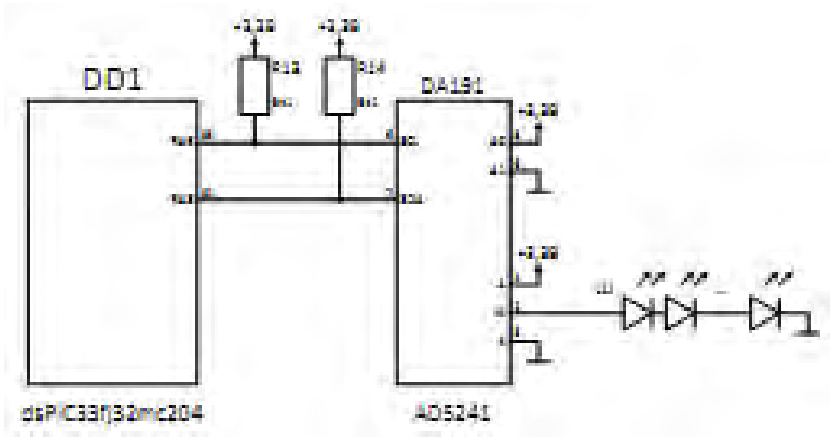



Рис. 2.1. Электрическая схема подключения микросхемы ЦАП

Линия тактирования SCL подключена к порту RA9 микроконтроллера, линия данных SDA – к порту RA8. Линии имеют подтягивающий резистор.

С учётом подключения выводов A и B микросхемы, напряжение на аналоговом выходе микросхемы будет рассчитываться по формуле:

Для визуального отображения уровня напряжения на аналоговом выходе к микросхеме ЦАП подключена линейная светодиодная шкала (LLI).

Адрес микросхемы ЦАП формируется из фиксированной части и битов, формируемых аппаратно в зависимости от схемы подключения дополнительных адресных входов микросхемы. Фиксированная часть представляет собой 01011 в двоичном коде, дополнительная часть адреса равна 01. Таким образом, при обращении к микросхеме ЦАП управляющее слово будет 01011010 (5A в шестнадцатеричном коде)

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, позволяющую индицировать на LLI уровень аналогового сигнала с регулятора RP1. Значение аналогового сигнала также дублировать на LED.

*Анализ задачи:* Операции по работе с интерфейсом $I^2C$ (чтение, запись данных, формирование стартовой и стопой последовательности) удобней организовать в виде соответствующих функций.

Листинг программы:

```
#include <P33FJ32MC204.h>
#define FOSC 7370000
#define FCY (FOSC / 2)
_FOSCSEL(FNOSC_FRC)      // настройка работы микроконтроллера

                         // от внутреннего тактового генератора

// объявление кодов сегментов

                    0x0
#define SEG_A       8
                    0x0
#define SEG_B       1
                    0x2
#define SEG_C       0
                    0x0
#define SEG_D       4
                    0x8
#define SEG_E       0
#define SEG_F       0x1
```

```
                        0
                        0x4
#define SEG_G           0
                        0x0
#define SEG_DP          2


1.      --- SEG_A ---
 //      |         |
 //  SEG_F    SEG_B
 //      |         |
//       --- SEG_G ---

 //      |         |
 //  SEG_E     SEG_C
 //      |         |
                --- SEG_D ---
 //                  SEG_DP

        // объявление кодов цифр
        #define N0          SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
        SEG_F
        #define N1          SEG_B + SEG_C

        #define N2          SEG_A + SEG_B + SEG_G + SEG_E + SEG_D
        #define N3          SEG_A + SEG_B + SEG_G + SEG_C + SEG_D
        #define N4          SEG_F + SEG_G + SEG_B + SEG_C

        #define N5          SEG_A + SEG_F + SEG_G + SEG_C + SEG_D

        #define N6          SEG_A + SEG_F + SEG_G + SEG_C + SEG_D +
        SEG_E
        #define N7          SEG_A + SEG_B + SEG_C
        #define N8          SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
        SEG_F + SEG_G
        #define N9          SEG_A + SEG_B + SEG_C + SEG_D + SEG_F +
        SEG_G

        #define DIP         SEG_DP
        #define MINUS       SEG_G

        #define OFF         0x00

        // массив кодов цифр
        char DIGITS[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};
```

```
                                        // номер активного вывода
char _i = 0;                            дешифратора
                                                  индикации - массив
char _ind[8];                           //  буфер  кодов
символов

// Инициализация таймера
T1
void
Init_Timer1()
{
                                    // сброс
        T1CON      = 0;             таймера
        IFS0bits.T1IF = 0;          // сброс флага прерывания таймера
        IEC0bits.T1IE = 1;          // разрешение прерывания от таймера
        TMR1       = 0x0000;        // обнуление текущего значения таймера
                              =
        PR1           0x0E65;       // задание периода таймера
                                    // разрешение работы таймера и его
        T1CONbits.TON = 1;          запуск
}

// Инициализация линий
портов
void Ind_Init()
{
        TRISBbits.TRISB6 =
        0;                                  // Выход    A     (RB6)
        TRISBbits.TRISB7 =
        0;                                  // Выход    B     (RB7)
        TRISBbits.TRISB8 =
        0;                                  // Выход    C     (RB8)
        TRISBbits.TRISB9 =
        0;                                  // Выход    D     (RB9)
        TRISCbits.TRISC3 =
        0;                                  // Выход    SDI   (RC3)
        TRISCbits.TRISC4 =                              SC
        0;                                  // Выход    K     (RC4)
        TRISBbits.TRISB5 =
        0;                                  // Выход    LE    (RB5)
}

// Отправление данных в регистр
MBI5168 void Ind_Send(char digit)
{
```

```
        char c;
        for (c = 0; c < 8; c++)

        {
                4.1     установка требуемого логического уровня

                4.2     на последовательном входе SDI
                драйвера MBI5168 if ((digit & (1 << c)) != 0)

                {
                        LATCbits.LATC3 = 1;
                }

                else
                {
                        LATCbits.LATC3 = 0;
                }
                4.3     формирование синхроимпульса на входе SCK

                LATCbits.LATC
                4       =       1;
                LATCbits.LATC
                4 = 0;

        }

        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
        LATBbits.LATB5 = 0;

}

5.1     Разбитие числа на
цифры void
Ind_Show(unsigned int b)
{

        _ind[7] = DIGITS[b % 10]; b /=
        10; _ind[6] = DIGITS[b % 10];
        b /= 10; _ind[5] = DIGITS[b %
        10];    b    /=    10;    _ind[4]    =
        DIGITS[b % 10];

}
```

5.2 Прерывание таймера T1 по совпадению

```
void __attribute__((interrupt,no_auto_psv)) _T1Interrupt()
{

        Ind_Send(OFF);                  // Отключение индикатора
                                        // Переключение на
        LATB &= ~(0x0F << 6);           следующий
        LATB |= (_i << 6);              // индикатор

        if (_i < 8)
        {
                Ind_Send(_ind[_i]);     // Отправка кода цифры
        }

        _i++;
        if (_i == 8)
        {
                _i = 0;
        }

                                        // Сброс флага прерывания
        IFS0bits.T1IF = 0;              таймера
        TMR
        1               = 0;            // Перезапуск таймера

}


#define Delay_us(d) (__delay32 (((d)) * ((FCY) / 1000000uL)))


                        TRISAbits.TRIS
#define SDA                         A8
                        TRISAbits.TRIS
#define SCL                         A9
#define READ_SDA   PORTAbits.RA8


// Инициализация портов I2C
void I2C_Init()
{
        ODCAbits.ODC
        A8                      = 1;        // Открытый коллектор SDA
        ODCAbits.ODC
        A9                      = 1;        // Открытый коллектор SCL
        LATAbits.LAT            = 0;
```

```
        A8
        LATAbits.LATA9        = 0;

        SDA = 1;

        SCL = 1;

}

// Формирование условия
СТАРТ void I2C_Start()

{
        SDA = 0;    Delay_us(25);

}


// Формирование условия
СТОП void I2C_Stop()
{

        SCL = 1;    Delay_us(25);
        SDA = 1;    Delay_us(25);
}

// Передача байта по
интерфейсу I2C char
I2C_Send(unsigned char c)

{
        char b;

        for (b = 7; b >= 0; b--)
        {
                SCL = 0;    Delay_us(25);

                if ((c & (1 << b)) != 0)
                {

                        SDA = 1;
                }
                else
                {
                        SDA = 0;
```

```
        }
        Delay_us(25);
        SCL = 1;    Delay_us(50);
    }
    1.  A
    CK
    SCL =
    0;
    2. Отпустить линию

    SDA = 1;    Delay_us(25);

    SCL = 1;    Delay_us(25);

    char result;

    if (READ_SDA == 0)
    {
        result = 0;
    }
    else
    {
        result = 1;

    }
                Delay_us(2
    SCL = 0;    5);
                Delay_us(2
    SDA = 0;    5);
    return
    result;
}

void ADC_Init()
{
    AD1CON1bits.AD12B
    = 0;                    // 10-битный режим
    AD1CON2bits.V
    CFG             = 0;    // Опорное напряжение – Avdd
    AD1CON1bits.SS          // Запуск преобразования после
    RC              = 0;    очистки
                            // бита выборки
    AD1PCFG
    L               = 0xFFFF;
```

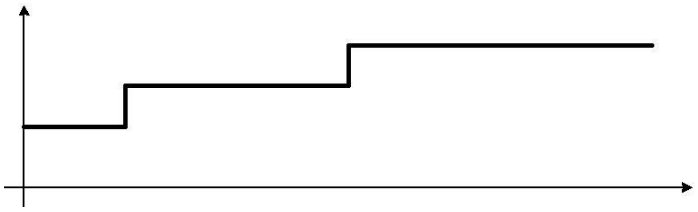
```
        AD1PCFGLbits.P
        CFG0            = 0;   // AN0 - аналоговый вход
        AD1PCFGLbits.P
        CFG1            = 0;   // AN1 - аналоговый вход
        AD1CON1bits.A           // Автоматический запуск выборки
        SAM             = 1;   после
                               // предыдущего преобразования
        AD1CON3bits.A           // Источник тактирования
        DRC             = 0;   преобразования -
                               // источник тактирования МК
        AD1CON1bits.F           // Формат данных - целый
        ORM             = 0;   беззнаковый
        AD1CON2bits.C
        HPS             = 1;   // Преобразование каналов CH0, CH1
        AD1CON1bits.SIMSA
        M= 1;                  // Одновременная выборка CH0, CH1

        AD1CHS123bits.CH123SA = 0;// Подключение входа AN0 на
                                  положительный // вход усилителя CH0

        AD1CHS0bits.CH0SA = 1; // Подключение входа AN1 на
                                  положительный // вход усилителя CH1

        AD1CON1bits.ADON  = 1;   // Включение модуля АЦП

}
void main()

{
        AD1PCFGL = 0xFFFF;          // Отключение входов АЦП
        I2C_Init();                 // Инициализация шины I2C
        ADC_Init();                 // Инициализация АЦП

        Ind_Init();                 // Инициализация индикации

        Init_Timer1();              // Инициализация таймера

        while (1)
        {
                AD1CON1bits.SAMP = 0;       // Запуск выборки
                while (!AD1CON1bits.DONE);        // Ожидание завершения
                                                  // преобразования

                unsigned int val = ADC1BUF1;
                Ind_Show(val);     // Отображение результата
```

```
            val = val / 4;          // Приведение результата к диапазону
            0..255

            // Отправление данных в ЦАП

            I2C_Start();
            I2C_Send(0x5
            A);
            I2C_Send(0x0
            0);
            I2C_Send(val)
            ; I2C_Stop();

            __delay32(FCY / 5);
        }

    }
```

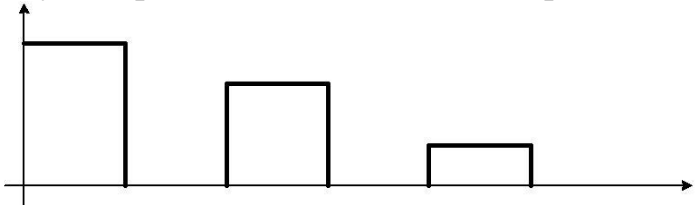
## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Отобразить на LLI количество включённых тумблеров VD1..VD10.
2. При любом изменении состояния тумблеров VD3..VD10 увеличивать количество включённых светодиодов на LLI.
3. При включении тумблера SA1 вывести на LLI периодичный сигнал вида:



4 При включении тумблера SA2 вывести на LLI периодичный сигнал вида:



5.При включении тумблера SA1 линейно увеличивать сигнал на LLI до максимального. При отключении тумблера SA1 – линейно уменьшать до ноля.

## 5 Контрольные вопросы

1. Какую функцию выполняют цифро-аналоговые преобразователи?
2. Какие применяются методы построения ЦАП?
3. Какого рода преобразование выполняет микросхема AD5241?
4. Из каких структурных элементов состоит микросхема AD5241?
5. Каким образом происходит управление микросхемой AD5241?

# Лабораторная работа 15

## Исследование технологии переназначаемых контактов и средств ввода квадратурного сигнала в микроконтроллер

**Цель работы:**

Изучить технологию переназначаемых контактов. Изучить алгоритм, способ настройки и принцип работы модуля квадратурного энкодера. Разработать и отладить программу для обработки квадратурного сигнала.

**Порядок выполнения работы:**

1. Изучить теоретические вопросы, связанные с технологией переназначаемых контактов и работой модуля квадратурного энкодера.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу системы ввода квадратурного сигнала.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

### 1 Краткие теоретические сведения

### 1.1 Переназначаемые контакты

Технология переназначаемых контактов позволяет периферийным модулям использовать не фиксированные, заранее определённые выводы микроконтроллера, а любые из линий ввода-вывода микроконтроллера, помеченных как RPx[5] (Remappable Pin). Таким образом, при программировании микроконтроллера существует возможность независимо сопоставлять входные и выходные линии периферийных модулей требуемым линиям портов. Функцией переназначаемых контактов могут воспользоваться следующие периферийные модули микроконтроллера:

1. 16-битный таймер;
2. модуль захвата/сравнения;
3. модуль квадратурного энкодера;
4. модуль приёмо-передатчика UART;
5. модуль внешних прерываний;
6. модуль SPI.

Соответствия переназначаемых контактов линиям ввода-вывода микроконтроллера настраиваются двумя группами регистров – для настройки входов и для настройки выходов периферийным модулей.

Для настройки функций входа используется группа регистров RPINRx[6].

Каждый регистр содержит набор 5-битных полей, ассоциированных с входами периферийных модулей. Значение поля определяет номер переназначаемого контакта, который будет использоваться данной функцией периферийного модуля. Использование переназначаемого контакта как входной линии периферийного модуля требует предварительную настройку используемой линии порта на вход с помощью установки соответствующего бита регистра TRISx[7].

Условная структура входов периферийных модулей, использующих технологию переназначаемых контактов, показана на рис. 1.1. Биты регистров RPINRx представлены в таблице 1.1.

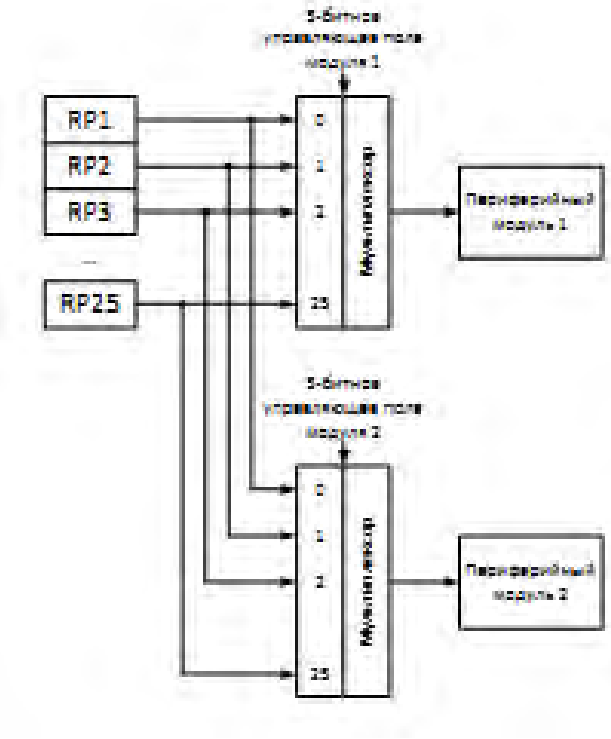



Рис. 1.1. Условная структура входов периферийных модулей

Таблица 1.1

Биты регистров RPINRx

| *Регистр* | *5-битное поле* | *Обозначение* | *Описание* |
|---|---|---|---|
| RPINR0 | INT1R<4:0> | INT1 | Вход внешнего прерывания |
| RPINR1 | INT2R<4:0> | INT2 | Вход внешнего прерывания |
| RPINR3 | T2CKR<4:0> | T2CK | Вход внешнего тактирования таймера Т2 |
| RPINR3 | T3CKR<4:0> | T3CK | Вход внешнего тактирования таймера Т3 |
| RPINR14 | QEA1R<4:0> | QEA | Вход сигнала А квадратурного энкодера |

Окончание таблицы 1.1

| | | | |
|---|---|---|---|
| RPINR14 | QEB1R<4:0> | QEB | Вход сигнала Б квадратурного энкодера |
| RPINR15 | INDX1R<4:0> | INDX | Вход сигнала индексной метки квадратурного энкодера |
| RPINR18 | U1RXR<4:0> | U1RX | Вход линии RX модуля UART |
| RPINR18 | U1CTS<4:0> | U1CTS | Вход линии TX модуля UART |
| RPINR20 | SDI1R<4:0> | SDI1 | Вход сигнала данных модуля SPI |
| RPINR20 | SCK1R<4:0> | SCK1 | Вход сигнала тактирования модуля SPI |
| RPINR20 | SS1R<4:0> | SS1 | Вход сигнала выбора ведомого модуля SPI |

Например, задание контакту RP2 (линии 2 порта RB) функции входа сигнала RX модуля UART осуществляется следующим образом:

```
TRISBbits.TRISB2          = 1;    // Настройка линии порта на вход
RPINR18bits.U1RXR         = 2;    // Задание в качестве входа RX модуля UART
                                  // контакта RP2
```

Для настройки функций выхода используется группа регистров RPORx[8]. Аналогично, каждый регистр содержит набор 5-битных полей, но каждое поле связано с одним переназначаемым контактом. Значение поля определяет функцию периферийного модуля, которая будет использовать данный вывод. Значение поля может быть так же нулевым для того, чтобы оставить контакт неподключенным ни к одному из выводов периферийных модулей.

Условная структура входов периферийных модулей, использующих технологию переназначаемых контактов, показана на рис. 1.2. Соответствие переназначаемых контактов и управляющих регистров приведено в таблице 1.2. Расшифровка функций периферийных модулей, использующих переназначаемые контакты в качестве выходов, представлена в таблице 1.3.

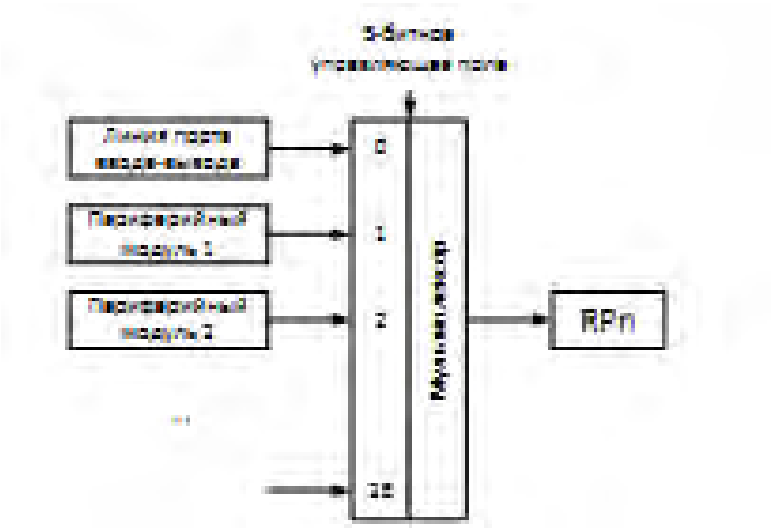



Рис. 1.2. Условная структура выводов периферийных модулей

Таблица 1.2

Соответствие переназначаемых контактов управляющим регистрам

| *Регистр* | RPOR0 | RPOR1 | RPOR2 | RPOR3 | RPOR4 | RPOR5 |
|---|---|---|---|---|---|---|
| *Выводы* | RP1, RP0 | RP3, RP2 | RP5, RP4 | RP7, RP6 | RP9, RP8 | RP11, RP10 |
| | | | | | | |
| *Регистр* | RPOR6 | RPOR7 | RPOR8 | RPOR9 | RPOR10 | RPOR11 |
| *Выводы* | RP13, RP12 | RP15, RP14 | RP16, RP17 | RP18, RP19 | RP20, RP21 | RP22, RP23 |
| | | | | | | |
| *Регистр* | RPOR12 | | | | | |
| *Выводы* | RP24, RP25 | | | | | |

Таблица 1.3

Функций периферийных модулей

| *Обозначение* | *Значение* | *Описание* |
|---|---|---|
| NULL | 00000 | Вывод PRn назначен линии порта ввода-вывода |
| U1TX | 00011 | Вывод PRn назначен выходу TX модуля UART |
| U1RTS | 00100 | Вывод PRn назначен выходу RTS модуля UART |
| SDO1 | 00111 | Вывод PRn назначен выходу данных модуля SPI |

Окончание таблицы 1.3

| | | |
|---|---|---|
| SCK1OUT | 01000 | Вывод PRn назначен выходу тактирования модуля SPI |
| SS1OUT | 01001 | Вывод PRn назначен выходу выбора ведомого модуля SPI |
| OC1 | 10010 | Вывод PRn назначен выходу модуля сравнения |
| OC2 | 10011 | Вывод PRn назначен выходу модуля сравнения |
| UPDN | 11011 | Вывод PRn назначен выходу направления модуля квадратурного энкодера |

## 1.2 Модуль квадратурного энкодера

Квадратурный сигнал - это два сигнала со сдвинутыми по отношению друг к другу на 90° прямоугольными импульсами (рис. 1.3). Для удобства работы сигналы обозначаются как правило A, B либо SIN, COS.

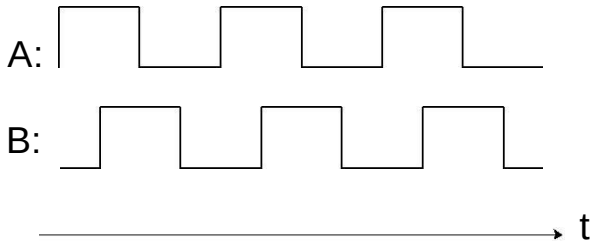



Рис. 1.3. Пример квадратурного сигнала

Наиболее распространённым источником квадратурного сигнала являются датчики углового перемещения, называемые так же инкрементальными энкодерами. Такие датчики применяются для определения положения и скорости вращения механизма и используются в системах замкнутого управления.

Счетом фронтов любой части квадратурного сигнала можно контролировать угловое движение, а по соотношению фаз сигналов - направление вращения. В случае, если сигнал фазы A опережает сигнал фазы B, направление вращения энкодера принимается за положительное. Если сигнал фазы A отстаёт от сигнала фазы B, направление вращения принимается за отрицательное.

Некоторые энкодеры могут дополнительно генерировать сигнал нулевой метки (т.н. индексный сигнал). Такой сигнал возникает один раз на полный оборот энкодера и применяется для определения абсолютного положения механизма.

Для обработки квадратурного сигнала микроконтроллер dsPIC33fj32mc204 имеет в своём составе модуль квадратурного энкодера, позволяющий аппаратно увеличивать либо уменьшать значение счётчика в зависимости от направления вращения подключённого энкодера. Модуль квадратурного энкодера включает:

1. 3 входных контакта для двух фаз квадратурного сигнала (QEA, QEB) и сигнала нулевой метки (INDX);
2. программируемые цифровые фильтры по входам сигналов;
3. квадратурный декодер, определяющий положение и направление вращения датчика; Модуль квадратурного энкодера может работать в двух режимах:
4. сброс счётчика положения происходит при достижении им заданного максимального значения (MAXCNT);
5. сброс счётчика положения происходит при появлении индексного сигнала.

   Во всех режимах возможно генерирование прерывания при сбросе счётчика положения.

Так же модуль может генерировать сигнал направления вращения энкодера (UPDN), определяемого по чередованию фаз квадратурного сигнала. Получаемый сигнал может быть скоммутирован на один из переназначаемых контактов.

Настройка модуля квадратурного энкодера осуществляется с помощью четырёх регистров:

1. QEICON – регистр управления и статуса;
2. DFLTCON – регистр настройки входных цифровых фильтров;

3. POSCNT – регистр счётчика положения;
4. MAXCNT – регистр максимального значения.

   Биты регистра QEICON указаны в таблице 1.4, их назначение – в таблице 1.5.

Таблица 1.4

Биты регистра QEICON

| *Обозначение* | CNTERR | - | QEISIDL | INDEX | UPDN | QEIM<2:0> | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | SWPAB | PDCOUT | TQGATE | TQCKPS<1:0> | | POSRES | TQCS | UDSRC |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.5

Назначение некоторых битов регистра QEICON

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15 | CNTERR | Флаг ошибки счётчика положения<br>CNTERR = 1: Произошла ошибка подсчёта положения<br>CNTERR = 0: Ошибка отсутствует |
| 11 | UPDN | Флаг направления вращения<br>UPDN = 1: Направление вращения положительное<br>UPDN = 0: Направление вращения отрицательное |
| 7 | SWPAB | Флаг обмена входов A и B<br>SWPAB = 1: Сигналы фаз A и B подключены прямо<br>SWPAB = 0: Сигналы фаз A и B перекрещены друг с другом |
| 4-3 | TQCKPS<1:0> | Предделитель счётчика положения<br>TQCKPS<1:0> = 11: Значение предделителя 256<br>TQCKPS<1:0> = 10: Значение предделителя 64<br>TQCKPS<1:0> = 01: Значение предделителя 8<br>TQCKPS<1:0> = 00: Значение предделителя 1 |
| 2 | POSRES | Источник сброса счётчика положения<br>POSRES = 1: Сигнал нулевой метки |

Таким образом, для настройки модуля квадратурного энкодера необходимо выполнить следующее:

1. Задать, какие из переназначаемых контактов будут использоваться в качестве входов квадратурного сигнала и нулевой метки (если используется);
2. Если требуется, задать предделитель счётчика положения, а так же настроить входные цифровые фильтры.

## 2 Электрическая принципиальная схема к лабораторной работе

На рис.2.1 приведена электрическая схема подключения источника квадратурного сигнала к микроконтроллеру.

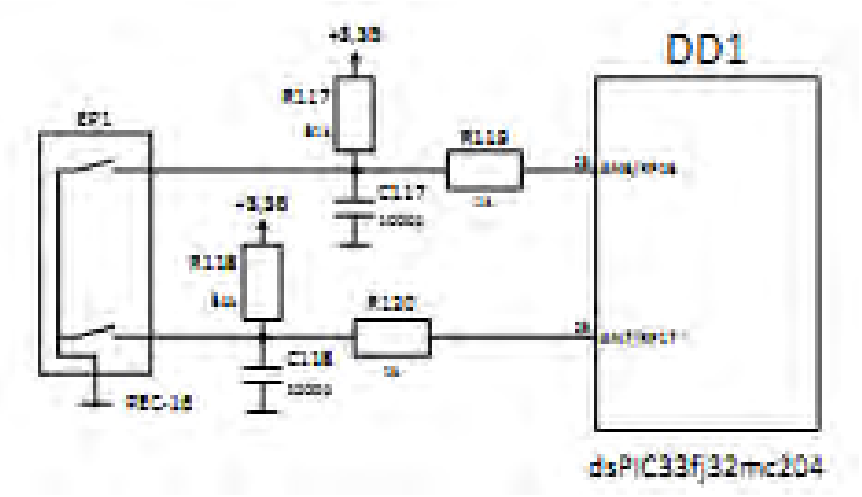



Рис. 2.1. Электрическая схема подключения источника квадратурного сигнала

В стенде в качестве источника квадратурного сигнала используется инкрементальный энкодер PEC-16, выводы которого через фильтрующие RC-цепочки подключены к входам RP16, PR17 микроконтроллера.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, отображающую на LED значение инкрементального энкодера EP1.

Листинг программы:

```
#include <P33FJ32MC204.h>
#define FOSC
7370000 #define
FCY (FOSC / 2)
_FOSCSEL(FNOSC_                    // настройка работы
FRC)                               микроконтроллера
                                   // от внутреннего тактового
                                   генератора

// объявление кодов сегментов
#define SEG_A          0x08        //     --- SEG_A ---
#define SEG_B          0x01        //       |           |
#define SEG_C          0x20        //   SEG_F     SEG_B
#define SEG_D          0x04        //       |           |
```

```
#define SEG_E           0x80        //    --- SEG_G ---
#define SEG_F           0x10        //      |           |
#define SEG_G           0x40        //  SEG_E    SEG_C
#define SEG_DP          0x02        //      |           |

                                    //       --- SEG_D ---   SEG_DP

// объявление кодов цифр
                        SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N0              SEG_F
#define N1              SEG_B + SEG_C
#define N2              SEG_A + SEG_B + SEG_G + SEG_E + SEG_D
#define N3              SEG_A + SEG_B + SEG_G + SEG_C + SEG_D
#define N4              SEG_F + SEG_G + SEG_B + SEG_C
#define N5              SEG_A + SEG_F + SEG_G + SEG_C + SEG_D
                        SEG_A + SEG_F + SEG_G + SEG_C + SEG_D +
#define N6              SEG_E
                        SEG_A + SEG_B +
#define N7              SEG_C
                        SEG_A + SEG_B + SEG_C + SEG_D + SEG_E +
#define N8              SEG_F + SEG_G
                        SEG_A + SEG_B + SEG_C + SEG_D + SEG_F +
#define N9              SEG_G

#define DIP             SEG_DP
#define
MINUS                   SEG_G

#define OFF             0x00

// массив кодов цифр
char DIGITS[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};
// Инициализация таймера
T1
    void Init_Timer1(
                void )
{
        T1CON = 0;              // сброс таймера
        IFS0bits.T1IF = 0;      // сброс флага прерывания таймера
                                // разрешение прерывания от
        IEC0bits.T1IE = 1;      таймера
        TMR1                    // обнуление текущего значения
        =       0x0000;         таймера
                0x0E65
        PR1 =           ;       // задание периода таймера
```

```
                                    // разрешение работы таймера и его
        T1CONbits.TON = 1;          запуск
}

char _ind = 0;          // номер активного индикатора

char _data[8];          // буфер индикации - массив кодов символов

// Отправление данных в регистр
MBI5168 void Ind_Send(char digit)

{
        char c;

        for (c = 0; c < 8; c++)

        {
                // установка требуемого логического уровня

                //      на      последовательном      входе      SDI
                драйвера MBI5168 if ((digit & (1 << c)) != 0)
                {
                        LATCbits.LATC3 = 1;
                }
                else
                {
                        LATCbits.LATC3 = 0;
                }
                //      формирование синхроимпульса на
                входе SCK LATCbits.LATC4 = 1;
                LATCbits.LATC4 = 0;
        }
        // формирование синхроимпульса на входе LE
        LATBbits.LATB5 = 1;
        LATBbits.LATB5 = 0;
}


// Инициализация линий портов
void Ind_Init()
{
        TRISBbits.TRISB6 = 0;               // Выход    A     (RB6)
        TRISBbits.TRISB7 = 0;               // Выход    B     (RB7)
        TRISBbits.TRISB8 = 0;               // Выход    C     (RB8)
        TRISBbits.TRISB9 = 0;               // Выход    D     (RB9)
```

```
        TRISCbits.TRISC3 = 0;           // Выход   SDI SC  (RC3)
        TRISCbits.TRISC4 = 0;           // Выход   K       (RC4)
        TRISBbits.TRISB5 = 0;           // Выход   LE      (RB5)
}


void Ind_Show(int pos)
{
        // Запоминание знака числа
        int negative = 0;
        if (pos < 0)
        {
                pos = -pos;
                negative = 1;

        }

        //      Очистка
        индикатора int i;

        for (i = 0; i < 7; i++)

        {

                _data[i] = OFF;

        }

        //      Разбиение числа на
        цифры int n = 7;

        do
        {
                _data[n] = DIGITS[pos %
                10]; pos /= 10;

                n--;

        } while (pos > 0);

        // Отображение знака числа

        if (negative)
```

```
        {
                _data[n] = MINUS;
        }
}
// Прерывание таймера T1 по совпадению
void __attribute__((interrupt,no_auto_psv)) _T1Interrupt( void )

{
        Ind_Send(OFF);          // Отключение индикатора
        LATB &= ~(0x07 <<       // Переключение на
        6);                     следующий
        LATB |= (_ind << 6);    // индикатор
        Ind_Send(_data[_ind])
        ;                       // Отправка кода цифры

        _ind++;                 // переход к след. индикатору
        if (_ind == 8)
        {
                _ind = 0;
        }
                                // Сброс флага прерывания
        IFS0bits.T1IF = 0;      таймера
        TMR
        1           = 0;        // Перезапуск таймера
}

// Инициализация модуля квадратурного
энкодера void Encoder_Init()

{
        TRISCbits.TRISC
        0                   = 1;        // Вход A
        TRISCbits.TRISC
        1                   = 1;        // Вход B
        RPINR14bits.QE                  // Задание выводу RP17 функции
        A1R                 = 17;       входа A
        RPINR14bits.QE                  // Задание выводу RP16 функции
        B1R                 = 16;       входа B
        QEICONbits.QEI
        M                   = 0b101;    // Режим работы модуля -
                                        // сброс по переполнению
}


void main()
```

```
{
        AD1PCFGL =
        0xFFFF;                         // Отключение выводов АЦП

        Encoder_Init();                 // Инициализация энкодера
        Ind_Init();                     // Инициализация индикации

        Init_Timer1();                  // Инициализация таймера


        while (1)
        {
                Ind_Show(POSCNT);       // Отображение результата
                __delay32(FCY / 10);
        }
}
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Вывести на LLI модуль скорости вращения энкодера EP1, VD1 использовать как индикатор знака скорости.
2. Вывести на LLI значение положения энкодера EP1. Превышение значения положения определённой величины сигнализировать миганием светодиода VD1.
3. Вывести на LED значение положения энкодера EP1. Превышение значения положения определённой величины сигнализировать миганием светодиода VD1.
4. Отобразить значение скорости вращения энкодера EP1 на VD1..VD10, используя светодиоды как линейную шкалу.
5. Вывести на LCD положение и скорость вращения энкодера.

## 5 Контрольные вопросы

1. В чём заключается технология переназначаемых контактов?
2. В каких случаях можно воспользоваться технологией переназначаемых контактов?
3. Каким образом происходит настройка функций переназначаемых контактов?
4. Что такое квадратурный сигнал?
5. Каким образом происходит обработка квадратурного сигнала?
6. Как настраивается модуль квадратурного энкодера микроконтроллера?

# Лабораторная работа 16

# Исследование интерфейса RS232

## Цель работы:

Изучить структуру и принцип работы последовательного интерфейса микроконтроллера. Разработать и отладить программу для обмена информации по последовательному каналу в заданном режиме.

## Порядок выполнения работы:

1. Изучить теоретические вопросы, связанные с технологией последовательного интерфейса.
2. Изучить принципиальную электрическую схему к лабораторной работе.
3. Разработать программу в соответствии с индивидуальным заданием.
4. Отладить программу в среде MPLAB IDE.
5. Загрузить программу в учебный стенд.
6. Исследовать работу системы ввода квадратурного сигнала.
7. Оформить отчёт по лабораторной работе.
8. Ответить на контрольные вопросы.

## 1 Краткие теоретические сведения

### 1.1 Интерфейс RS232

Интерфейс RS232 разработан ассоциацией электронной промышленности как стандарт для соединения компьютеров и различных последовательных периферийных устройств.

Последовательная передача данных означает, что данные передаются по единственной линии. При этом биты данных передаются по очереди с использованием одного провода. Уровень напряжения для логического ноля составляет -15..-3В, для логической единицы – +3..+15В. Промежуток -3..+3В соответствует неопределённому значению.

Для синхронизации группе битов данных обычно предшествует специальный стартовый бит, после группы битов следуют бит проверки на чётность (иногда может отсутствовать) и один либо два стоповых бита (рис. 1.1).


Рис. 1.1. Формат передачи данных интерфейса RS232

Из рисунка видно, что исходное состояние линии последовательной передачи данных – уровень логической 1. Это состояние линии называют отмеченным – MARK. Когда начинается передача данных, уровень линии переходит в 0. Это состояние линии называется пустым – SPACE.

Стартовый бит START сигнализирует о начале передачи данных. Далее передаются биты данных (вначале младшие, затем старшие).

Затем передаётся бит чётности. Бит чётности имеет такое значение, чтобы в пакете битов общее количество единиц было чётно либо нечётно, в зависимости от настройки параметров связи. Этот бит служит для обнаружения ошибок, которые могут возникнуть при передаче. Приёмное устройство заново вычисляет чётность данных и сравнивает результат с принятым битом чётности. Если чётность не совпала, то считается, что данные переданы с ошибкой.

- конце посылки передаются один или два стоповых бита STOP, сигнализирующие
- завершении передачи. Затем до прихода следующего стартового бита линия снова переходит в состояние MARK.

Использование бита чётности, стартовых и стоповых битов определяют формат передачи данных. Очевидно, что передатчик и приёмник должны использовать один и тот же формат данных.

Другая важная характеристика – скорость передачи данных. Она так же должна быть одинаковой для передатчика и приёмника. Скорость передачи данных измеряется в бодах (бит в секунду).

- самом простейшем случае интерфейс RS232 требует 2 сигнальные линии – TD (transmit data) и RD (received data). Линия TD используется для передачи данных, линия RX
- для приёма. Таким образом, при соединении двух устройств, необходимо

линию TD первого подключить на линию RD второго, и линию TD второго подключить на линию RD первого.

### 1.2 Модуль последовательного интерфейса микроконтроллера

Модуль последовательного интерфейса микроконтроллеров семейства dsPIC33 (UART) является асинхронным дуплексным интерфейсом с расширенной функциональностью и имеет следующие преимущества:

1. увеличенная скорость передачи;
2. аппаратный контроль чётности;
3. тактовый генератор модуля имеет 16-битный предделитель, который обеспечивает установку скорости обмена данных в широких пределах с высокой точностью;
4. 4-уровневый буфер приёмника и передатчика;
5. аппаратное управление направлением передачи данных;
6. реализация физического уровня IrDA.


Рис. 1.1. Упрощенная схема модуля асинхронного обмена данными

### 1.3 Аппаратно-программная архитектура модуля последовательного интерфейса микроконтроллера

Для управления параметрами обмена данными, а так же контроля передачи и приёма данных по последовательному интерфейсу используется следующая группа регистров:

1. U1MODE – регистр установки режимов работы;
2. U1STA – регистр управления/состояния порта;
3. U1RXREG – регистр-приёмник данных;
4. U1TXREG – регистр передачи данных;
5. U1BRG – регистр установки скорости обмена.

Биты управляющих регистров и их назначение представлены в таблицах 1.1-1.10.

Таблица 1.1

Биты регистра U1MODE

| *Обозначение* | UARTEN | - | USIDL | IREN | RTSMD | - | UEN<1:0> | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | WAKE | LPBACK | ABAUD | URXINV | BRGH | PDSEL<1:0> | | STSEL |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.2

Назначение некоторых битов регистра U1MODE

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 15 | UARTEN | Разрешение работы порта<br>UARTEN = 1: Работа порта разрешена<br>UARTEN = 0: Работа порта запрещена |
| 9-8 | UEN<1:0> | Управление выводами портов<br>UEN = 11: Выводы U1TX, U1RX, BCLK1 включены и используются<br>UEN = 10: Выводы U1TX, U1RX, ~U1CTS, ~U1RTS включены и используются<br>UEN = 01: Выводы U1TX, U1RX, ~U1RTS включены и используются<br>UEN = 00: Выводы U1TX, U1RX включены и используются |
| 3 | BRGH | Установка скорости обмена порта<br>BRGH = 1: Работа порта на повышенной скорости<br>BRGH = 0: Работа порта на обычной скорости |

Окончание таблицы 1.2

| | | |
|---|---|---|
| 2-1 | PDSEL<1:0> | Выбор формата данных и чётности<br>PDSEL = 11: 9-битные данные, без бита чётности<br>PDSEL = 10: 8-битные данные, проверка на нечётность<br>PDSEL = 01: 8-битные данные, проверка на чётность<br>PDSEL = 00: 8-битные данные, без бита чётности |
| 0 | STSEL | Выбор количества стоповых бит<br>STSEL = 1: 2 стоповых бита<br>STSEL = 0: 1 стоповый бит |

Таблица 1.3

Биты регистра U1STA

| *Обозначение* | UTXISEL1 | UTXINV | UTXISEL0 | - | UTXBRK | UTXEN | UTXBF | TRMIT |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | URXISEL<1:0> | | ADDEN | RIDLE | PERR | FERR | OERR | URXDA |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.4

Назначение некоторых битов регистра U1STA

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 10 | UTXEN | Разрешение передачи данных<br>UTXEN = 1: Передача данных разрешена, вывод U1TX контролируется модулем последовательного интерфейса.<br>UTXEN = 0: Передача данных запрещена. |

Окончание таблицы 1.4

| | | |
|---|---|---|
| 9 | UTXBF | Состояние буфера передатчика |
| | | UTXBF = 0: Буфер передатчика не заполнен |
| 0 | URXDA | Состояние буфера приёмника<br>UTXBF = 1: Буфер приёмника содержит данные<br>UTXBF = 0: Буфер приёмника пуст |

Таблица 1.5

Биты регистра U1RXREG

| *Обозначение* | - | - | - | - | - | - | - | URX8 |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | URX<7:0> | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.6

Назначение некоторых битов регистра U1RXREG

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 8 | URX 8 | 8-й бит принятых данных (в режиме 9-битных данных) |
| 7-0 | URX<7:0> | Биты 7-0 принятых данных |

Таблица 1.7

Биты регистра U1TXREG

| *Обозначение* | - | - | - | - | - | - | - | UTX8 |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

| *Обозначение* | UTX<7:0> | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Номер бита* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Таблица 1.8

Назначение некоторых битов регистра U1TXREG

| *Номер бита* | *Обозначение* | *Описание* |
|---|---|---|
| 8 | UTX8 | 8-й бит передаваемых данных (в режиме 9-битных данных) |
| 7-0 | UTX<7:0> | Биты 7-0 передаваемых данных |

## 2 Электрическая схема к лабораторной работе

На рис.2.1 приведена электрическая схема к лабораторной работе.

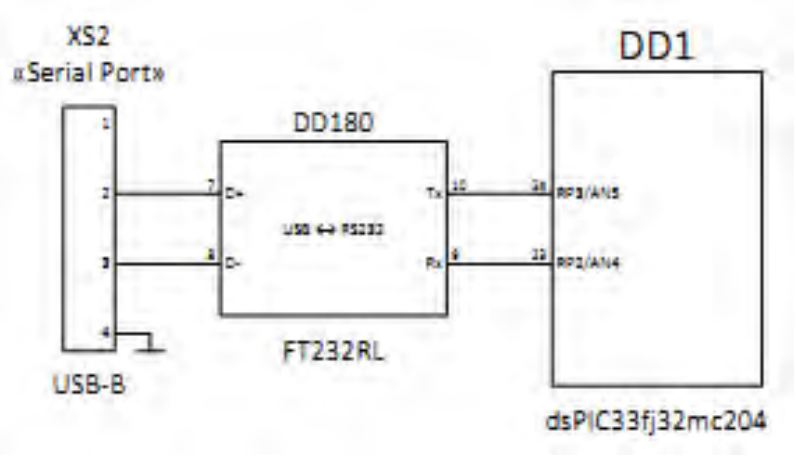



Рис. 2.1. Электрическая схема к лабораторной работе

В стенде связь микроконтроллера с внешним устройством осуществляется по интерфейсу RS232. Для возможности подключения к компьютерам, имеющим более современную шину – шину USB, в стенде применена микросхема FT232RL. Данная микросхема абсолютно прозрачно для пользователя преобразует поток данных последовательного интерфейса в поток данных шины USB, позволяя работать с устройством через виртуальный COM-порт аналогично реальному COM-порту.

## 3 Пример выполнения работы

*Задача:* Разработать программу для учебного стенда, принимающую и отправляющую в ответ принятые данные по последовательному интерфейсу RS232 от внешнего устройства.

Листинг программы:

```
#include <P33FJ32MC204.h>
```

```
#define FOSC 7370000
#define FCY (FOSC / 2)
_FOSCSEL(FNOSC_F                    // настройка работы
RC)                                 микроконтроллера
                                    // от внутреннего тактового
                                    генератора
#define BAUDRATE
9600                                        // Задание скорости обмена
#define BRGVAL
(((FCY/BAUDRATE)/16)-1)                     // Расчѐт предделителя


void main()
{
        AD1PCFGL =
        0xff;                       // Отключение входов АЦП
        TRISBbits.TRISB
        2               = 1;        // Вход RX
        RPINR18bits.U1              // Задание выводу RP2 функции
        RXR             = 2;        входа RX
                                    // Задание выводу RP3 функции
        RPOR1bits.RP3R  = 3;        выхода TX
        U1MODEbits.ST
        SEL             = 0;        // 1 стоп-бит
        U1MODEbits.PD               // 8-битные данные, без бита
        SEL             = 0b00;     чѐтности
        U1MODEbits.BR               // Работа модуля на обычной
        GH              = 0;        скорости
                        =
        U1BRG           BRGVAL;     // Установка требуемой скорости

        U1MODEbits.UA
        RTEN            = 1;        // Включить модуль UART
        U1STAbits.UTXE
        N               = 1;        // Разрешение передачи данных


        while (1)
        {
                if (U1STAbits.URXDA
                == 1)                       // Проверка буфера приѐмника
                                            // на наличие данных
                {
```

```
                char data =
                U1RXREG;
                U1TXREG = data;

            }

        }

    }
```

## 4 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую выполнить следующие действия:

1. Вывести на LCD принятые по последовательному интерфейсу данные. После каждого полученного байта данных передать внешнему устройству количество принятых данных.
2. По запросу внешнего устройства передать ему информацию о состоянии тумблеров SA2..SA10.
3. Отобразить принятые по последовательному интерфейсу данные на LED. В случае приёма определённой последовательности данных отправить внешнему устройству соответствующее сообщение.
4. Периодически производить отправку сообщений по последовательному интерфейсу. На LED вывести количество принятых ответных сообщений.

## 5 Контрольные вопросы

1. Каким образом происходит последовательная передача данных?
2. Для чего предназначен модуль последовательной передачи данных микроконтроллера?
3. Каким образом изменить скорость передачи данных последовательного порта?
4. Каким образом задаётся режим работы модуля последовательной передачи данных микроконтроллера?

# 4.**ПРОГРАММИРОВАНИЕ УЧПУ**

Написать коментарий к каждой строке следующих программ обработки деталей на станках с УЧПУ:

-линейная нтерполяция;

```
%0002
N0010  G54 G92 X0.000 Z50.000
N0020  G59
N0030  G00 X60 Z60
N0040  T0101
N0050  G00 X20.000 Z0.000
N0060  G96 S120 M04
N0070  G01 X-1.000 Z0.000 F200
N0080  G00 X14.000 Z1.000
N0090  G01 X20.000 Z-25.000
N0100  G00 X50.000 Z60.000
N0110  G53 G56 T0000 M30
```

-круговая интерполяция ;

```
%0001
N0010  G54 G92 X0.000 Z50.000
N0020  G59
N0030  G00 X60.000 Z60.000
N0040  T0303
N0050  G00 X40.000 Z0.000
N0060  G02 X40.000 Z-20.000 I10.000 K-10.000
N0070  G00 X60.000 Z60.000
N0080  G53 G56 T0000 M30
%0003
N0010  G54 G92 X0.000 Z50.000
N0020  G59
N0030  G00 X60.000 Z60.000
N0040  T0303
N0050  G00 X40.000 Z-20.000
N0060  G03 X40.000 Z0.000 I10.000 K10.000
N0070  G00 X60.000 Z60.000
N0080  G53 G56 T0000 M30
```

-многопроходной цикл:

```
N0010  G54 G92 X0.000 Z50.000
N0020  G59
N0030  G00 X60.000 Z60.000
N0040  T0101
N0050  G96 S120 F200 M03
```

```
N0060  G00 X42.000 Z2.000
N0070  G84 X16.000 Z-30.000 D3=2000 D0=1000 D2=1000
N0080  G00 X60.000 Z60.000
N0090  G53 G56 T0000 M30
```

-контурная обработка:

```
%
00001
(PROGRAM NAME - CONTOUR 1) N100 G21
N102  GO G17 G40 G49 G80 G90 (FREZA D5)
N104  T1 M6
N106  GO G90 G54 X25. Y-27.5 S2000 M3
N108  G43 HI Z100.
N110  Z10.
N112  G1 Z-4. F100.
N116  X-27.5
N118  Y20.
N120  G2 X-20. Y27.5 R7.5 N122 G1X1.036 N124 X27.5 Y1.036 N126 Y-20.
N128  G2 X20. Y-27.5 R7.5
N130  G1Z6.
N132  GO Z100.
N134  M5
N136  G91G28 ZO.
N138  G28 XO. YO.
N140  M30

%
00002
(PROGRAM NAME - CONTOUR2 ) N100 G21
N102  GO G17 G40 G49 G80 G90 ( FREZA D5 )
N104  T1 M6
N106  GO G90 G54 X25. Y-35. S2000 M3
N108  G43H1Z100.
N110  Z10.
N112  G1 Z-4.F100.
N114  G41 D1 Y-30.
N116  G3 X20. Y-25. R5.
N118  Gl X-25.
N120  Y20.
N122  G2 X-20. Y25. R5.
N124  Gl X0.
N126  X25. Y0.
N128  Y-20.
N130  G2 X20. Y-25. R5.
```

N132 G3 X15. Y-30. R5.

N134 GI G40 Y-35.
N136 Z6.
N138 G0Z100.
N140 M5
N142 G91 G28Z0.
N144 G28 XO. Y0.
N146 M30
%

-обработка карманов:
%
00003
(PROGRAM NAME - FINISH POCKET) N100 G21
N102 GO G17 G40 G49 G80 G90 ( FREZA D5)
N104 T1 M6
N106 GO G90 G54 X-2.5 Y-2.5 S1000 M3
N108 G43H1Z100.
N110 Z10.
N112 G1 Z-2 FI00.
N114 Y-5.
N116 G3X0 Y-7.5 R2.5
N118 G1 X10
N120 G3 X17.5 YO. R7.5
N122 X10. Y7.5 R7.5
N124 G1X-10
N126 G3 X-17.5 YO. R7.5
N128 X-10. Y-7.5 R7.5
N130 G1 XO.
N132 G3 X2.5 Y-5. R2.5
N134 G1 Y-2.5
N136 Z8.
N138 GO Z100.
N140 M5
N146 M30

00005
(PROGRAM NAME - ROUGH POCKET) N100 G21
N102 GO G17 G40 G49 G80 G90 N104T1 M6
N106 GO G54 X-13 75 Y3.75 S1000 M3
N108 G43H1Z100
N110 Z10.
N112 GI Z-I. F100.

```
N114  Y-3.75
N116  X13.75
N118  Y3.75
N120  X-13.75
N122  X-17.5 Y7.5
N124  Y-7.5
N126  X17.5
N128  Y7.5
N130  X-17.5
N132  X-25. Y15.
N134  Y-15.
N136  X25.
N138  Y15.
N140  X-25.
N142  Z9.
N144  GO Z100.
N146  M5
N152  M30

%
00006
(PROGRAM NAME - N6 )
N100  G21
N102  GO G17 G40 G49 G80 G90
N104  T1 M6
N106  GO G90 G54 X0. Y0. S1000 M3
N108  G43H1Z100.
N110  Z10.
N112  G1 Z-.5 F100.
N120  X5. F200
N122  G3 X-5. R5.
N124  X5. R5.
N126  G1 X10.
N128  G3 X-10. R10.
N130  X10. R10.
N132  G1X15.
N134  G3 X-15. R15.
N136  X15.R15.
N138  G1Z10F300.
N140  G0Z100.
N142  M5
N148  M3O
%
```

**Ознакомиться с принципами программирования обработки УЧПУ Sinumerik 840D и написать коментарий к программе**

Программа обработки деталей состоит из последовательности кадров ЧПУ. Каждый кадр представляет собой один шаг обработки. В кадре записываются операторы в форме слов. Последний кадр содержит специальное слово для конца программы: M2, M17 или M30.

**Имена программ**: Каждая программа имеет собственное имя (max 31 знак) и не содержит пробелы. Имя выбирается свободно (можно использовать символ подчёркивания) с соблюдением следующих условий (кроме формата перфоленты):

- Первыми двумя символами должны быть буквы (также и буква с символом подчёркивания)
- Прочие цифры и буквы
  Пример: _MPF100 или WELLE, или WELLE_2

Имена файлов (формат перфоленты): имена файлов могут включать знаки 0…9, A…Z, a…z или _ и иметь максимальную длину в 24 знака. Имена файлов должны иметь 3-х буквенное расширение (xxx). Данные в формате перфоленты могут создаваться отдельно или обрабатываются в редакторе. Имя файла, сохраненного в памяти ЧПУ, начинается с "_N_". Файл в формате перфоленты вводятся в следующей последовательности: %<имя>. Символ % должен стоять в первой графе первой строки.

Пример: %_N_WELL123_MPF – программа обработки детали с именем WELLE123

Для создания программ имеются следующие символы: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, Q, S, T, U, V, W, X, Y, Z. При этом не путать букву "O" с числом "0". Прописные буквы не различаются. Специальные символы:

| | |
|---|---|
| % | Символ начала программы (только для создания программы на внешнем PC) |
| (…) | Заключение в скобки параметров |
| […] | Заключение в скобки адресов или индексов поля |
| < > | Меньше или больше |
| : | Главный кадр, конец метки, связывающий оператор |
| = | Присвоение, часть равенства |
| / | Деление, пропуск |
| * | Умножение |
| + | Сложение |
| - | Вычитание, отрицательный знак |
| " | Кавычки, идентификация для цепочки символов |
| _ | Символ подчеркивания, относительно к буквам |
| . | Десятичная точка |
| , | Запятая, знак разделения параметров |
| ; | Начало комментария |
| пробел | Знак разделения |
| LF | Конец кадра |

Программа ЧПУ состоит из отдельных кадров, кадр из слов и он должен включать в себя все данные для выполнения рабочей операции, и заканчивается символом LF (LINE FEED - новая строка). Символ LF не записывается, он создаётся автоматически при переключении строк. Кадр может состоять max из 512 символов, включая комментарий и символ конца кадра. Обычно в актуальной индикации кадра на дисплее показываются три кадра max по 66 символов каждый, включая и комментарии. Сообщения показываются в отдельном окне сообщений. Для наглядности структуры кадра, слова должны располагаться следующим образом, например, N10 G X Y Z F S T D M H. Некоторые адреса могут использоваться многократно в одном кадре, например: G, M, H. Различают два вида кадров: главные кадры и вспомогательные кадры. В главном кадре должны быть указаны все слова, необходимые для запуска технологического цикла. Главные кадры могут находятся как в основной программе, так и в подпрограммах. СЧПУ не проверят, содержит ли главный кадр всю необходимую информацию. Последовательность нумерации кадров может быть любой, но рекомендуется растущая последовательность номеров. Можно программировать кадры и без номеров. Последовательность записи в кадре произвольная, целое число отделяется от дробной части точки и записывается сразу за названием адреса без пробела. Информационные слова отделяются дуг от друга пробелом, дискретность задания 0,001мм. При расширенном использовании адреса применяется разделитель «=», например S1=470 (число оборотов для первого шпинделя) или X=AC(40) и т.д. Расширенное написание допускается только для следующих простых адресов: X, Y, Z, I, J, K, S, SPOS (SPOSA), M, H, T, F. Кадры, которые должны быть пропущены, обозначаются символом «/» перед номером кадра, например: /N10… или /0N10… - первый уровень пропуска, /1N10… - второй уровень пропуска и т.д. , всего до 10 уровней, в зависимости от версии программного обеспечения. Чтобы сделать УП понятной, рекомендуются вставлять в программу комментарии (связанные по смыслу), которые располагаются в конце кадра и обозначаются символом «;», например: N5 G54 T2 D1 M6 S400 M3; *деталь №125-65.13 для насоса типа TP23*. Вставленные комментарии можно использовать при поиске кадра. Они сохраняются в тексте УП и появляются в ходе выполнения программы в отображении текущего кадра. Можно также использовать пользовательские сообщения, которые программируются в УП для того, чтобы оператор во время выполнения программы получал информацию о текущей ситуации в процессе обработки детали. Сообщение отображается на экране до тех пор, пока оно не бедует заменено на новое или отменено. Текст сообщения может быть длиной максимум 124 знака и показывается в двух стоках по 62 символа в каждой строке. Сообщение в программе создается по средствам записи после кодового слова “MSG” в круглых скобках и текста сообщения в кавычках. Сообщение может быть отменено через “MSG ()”. Пример: N10 MSG (“черновая обработка контура”), N90 MSG ( )-отменить сообщение из кадра N10. Внутри программы можно запрограммировать ответвления с помощью

меток и безусловных переходов, т.е. последовательность обработки в программе может быть изменена. Имена меток задаются min с 2-мя и max с 32-мя знаками (буквы, цифры, символ подчеркивания). Первыми двумя знаками должны быть буквы или символы подчеркивания. После имени метки следует двоеточие ":". Метки должны быть уникальными в пределах программы и всегда стоят в начале кадра после номера. Команды безусловных переходов (целью являются метки или номера кадров):

GOTOB – оператор перехода в направлении начала программы

GOTOF – оператор перехода в направлении конца программы

GOTO – оператор перехода с поиском, сначала к концу программы и только потом к началу УП

Примеры задания перехода:

N15 ANF: G0 X150 Y150 - метка с именем ANF

N50 GOTOB ANF – команда перехода на кадр N15

N70 GOTOF N100 – команда перехода на кадр N100

Безусловный переход должен быть запрограммирован в отдельном кадре. Для программ с безусловными переходами команда M2(M30) не обязательно должна стоять в конце программы.

**Повторение части программы.** В отличае от подрограмм повторение части программы позволяет внутри одной программы повторять уже написанные части программы в любом составе. При этом по средствам меток обозначаются кадры или сегменты программы, которые должны быть повторены.

Операторы:

REPEATB P=… - повторение кадра (если P не указан, то кадр повторяется один раз), где P – число повторений. После последнего повторения программы продолжается с кадра следующего за строкой REPEATB. Обозначенный с помощью метки кадр может стоять до или после оператора REPEATB. Поиск изначально осуществляется в направлении начала программы, если метка в этом направлении не найдена, то поиск осуществляется в направлении конца программы.

REPEAT P=… - повторение группы кадров между меткой и оператором, если кадр с меткой содержит другие операторы, то они заново выполняются при каждом повторении. После последнего повторения программа продолжается с кадра следующего за строкой REPEAT. Метка должна стоять перед оператором REPEAT. Поиск осуществляется только в направлении начала программы.

1. Пример повторения кадров:

   N10 POS1: X10 Y20

   N20 POS2: CYCLE 81 (10, 0, 2, -18); цикл сверления

   …..

   N40 REPEATB POS1 P=5 – выполнить кадр N10 5 раз

   N50 REPEATB POS2 – выполнить кадр N20 один раз

   N60…

2. Пример повторения группы кадров:
N5…
N10 BEGIN: X… Y…
…..
N80 REPEAT BEGIN P=2 – повторение с N10 до N80 2 раза
N90…
3. Пример повторения сегмента (между метками) от BEGIN до END
N5…
N10 BEGIN: X… Y…
…..
N70 END: Z=10
…..
N100 REPEAT BEGIN END P=3 – выполнить диапазон N10 – N70 3 раза

Если стартовая метка BEGIN найдена перед оператором REPEAT, а конечная метка END за оператором REPEAT, то повторение осуществляется между стартовой меткой и оператором REPEAT.

**Адреса, G, M – функции, циклы**

| Адрес | Значение (стандартная установка) | Примечание |
|---|---|---|
| N | Номер кадра | Фиксированный |
| G | Функция перемещения | Фиксированный |
| M | Вспомогательная функция | Фиксированный |
| F | Подача | Фиксированный |
| S | Число оборотов шпинделя | Фиксированный |
| T | Номер инструмента | Фиксированный |
| D | Номер режущей кромки | Фиксированный |
| L | Вызов подпрограммы | Фиксированный |
| P | Кол-во повторения | Фиксированный |
| R | R-параметр (0-99) | Фиксированный |
| X Y Z<br>XYZ=AC(…)<br>XYZ=JC(…) | Оси<br>Абсолютное значение<br>Инкрементальное задание | Устанавливаемый |
| I J K | Параметры интерполяции | Устанавливаемый |
| CHR=…<br>CHF=… | Снятие фасок угла контура по ширине<br>Снятие фасок угла контура по длине | Фиксированный |
| RND<br>RNDM | Закругление угла контура<br>Закругление угла контура (модально) | Фиксированный |
| CR=… | Радиус окружности | Устанавливаемый |

| Функция | Значение | Действие m/s |
|---|---|---|
| G0 | Движение ускоренного хода | M |
| G1 | Линейная интерполяция | M |
| G2-G3 | Круговая интерполяция по/против часовой стрелке | M |
| CIP | Круговая интерполяция через промежуточную точку | M |
| G4 | Время ожидания (пауза) | S |
| G9 | Уменьшение скорости, точный останов | S |
| G17 | Выбор плоскости 1-ая/2-ая геометрическая ось X/Y | M |
| G18 | Выбор плоскости 3-ая/1-ая геометрическая ось Z/X | M |
| G19 | Выбор плоскости 2-ая/3-ая геометрическая ось Y/Z | M |
| G40 | Нет коррекции радиуса инструмента | M |
| G41 | Коррекция радиуса инструмента слева от контура | M |
| G42 | Коррекция радиуса инструмента справа от контура | M |
| G53 | Покадровое отключение G54…G599 | S |
| G54-G57 | Устанавливаемое смещение нулевой точки | M |
| G64 | Режим управления траекторией | M |
| G70 | Дюймовое указание размеров (длины) | M |
| G71 | Метрическое указание размеров (длины) | M |
| G90 | Абсолютное указание размеров | M |
| G91 | Относительное указание размеров | M |
| G94 | Линейная подача в мм/мин, дюймах/мин | M |
| G95 | Оборотная подача в мм/об дюймах/мин | M |
| G96 | Постоянная скорость резания | M |
| G97 | Отмена G96 | M |
| G147 | Мягкий подвод по прямой | S |
| G148 | Мягкий отвод по прямой | S |
| G247 | Мягкий подвод по четверти круга | S |
| G248 | Мягкий отвод по четверти круга | S |
| G347 | Мягкий подвод по полуокружности | S |
| G348 | Мягкий отвод по полуокружности | S |
| G450 | Переходная окружность, инструмент обходит углы детали по круговой траектории с R-инструмента | M |
| G451 | Точка пересечения эквидистант, свободное резание инструмента в углу детали | M |

| № п./п. | Функции | Назначения |
|---|---|---|
| 1 | M00 | Останов программы (программируемый) |
| 2 | M01 | Останов программы (с подтверждением) |
| 3 | M02 | Конец программы (без возврата в начало УП) |
| 4 | M03 | Вращение шпинделя по часовой стрелке |
| 5 | M04 | Вращение шпинделя против часовой стрелки |
| 6 | M05 | Останов шпинделя без ориентирования |
| 7 | M06 | Смена инструмента (программируется только в цикле L06) |
| 8 | M08 | Включение СОЖ |
| 9 | M09 | Выключение СОЖ |
| 10 | M17 | Конец подпрограммы |
| 11 | M30 | Конец программы |

| Цикл | Назначение | Цикл | Назначение |
|---|---|---|---|
| CYCLE81 | Сверление/центрование | CYCLE88 | Расточка 4 |
| CYCLE82 | Сверление/цекование | CYCLE89 | Расточка 5 |
| CYCLE83 | Глубокое сверление | HOLES1 | Ряд отверстий |
| CYCLE84 | Резьба метчиком без компенсирующего патрона | HOLES2 | Окружность отверстий |
| CYCLE840 | Резьба метчиком с компенсирующим патроном | CYCLE801 | Решётка отверстий |
| CYCLE85 | Расточка 1 | CYCLE 72 | Фрезерование контура |
| CYCLE86 | Расточка 2 | SLOT1,2 | Пазы на окружности Кольцевая канавка |
| CYCLE87 | Расточка 3 | POCKET3,4 | Прямоугольный карман Круговой карман |

## Системы отсчёта, программирование S, F, G4, перемещений, управление траекторией

1. **Ввод абсолютного размера.**

Формат: G90 или X=AC(…) Y=AC(…) Z=AC(…)

С помощью команды G90 или покадровым указанием AC (Absolute Count) определяется подвод отдельных осей к заданным позициям в абсолютном размере. Команда G90 действует модально (до отмены), AC только на кадр и действует для всех указанных осей, а также для позиционирований шпинделя SPOS, SPOSA и параметров интерполяции I, J, K (G90/G91 не влияют на неё).

Пример: N30 G2 X20 Y30 I=AC(45) J=AC(35) – центр окружности задан в абсолютном размере.

2. **Ввод относительного размера**

Формат: G91 или X=IC(…) Y=IC(…) Z=IC(…)

С помощью команды G91 или покадровым указанием IC (Incremental Count) определяется подвод отдельных осей к заданным позициям в относительном размере. Команда G91 действует модально (до отмены), IC только на кадр и действует для всех указанных осей, а также для позиционирований шпинделя SPOS, SPOSA и параметров интерполяции I, J, K (G90/G91 не влияют на неё).

Пример: N30 G2 X20 Y30 I0 J-25 – центр окружности задан в относительном размере.

3. **Программирование** S, F, G4

Для процесса обработки необходимо установить правильную скорость резания, значение подачи, а также определить число оборотов шпинделя и направление его вращения. Программирование:

N10 S1500 M3 S2=2500 M2=3 – число оборотов и направление вращения для главного шпинделя и для второго шпинделя (расширенное задание адреса).

N30 M19 или M1=19 – ориентированный останов главного шпинделя

Подача по траектории программируется адресом F (Feedrate) и действует до тех пор, пока не будет задано новое значение подачи и действует только в сочетании с соответствующей G – командой. После адреса F допускаются разделительные символы, например: F150 или F=150. С помощью функции FB можно задать подачу для отдельного кадра, на последующие кадры она не действует. Так же она не действует, если в кадре не задано перемещение.

Пример:

N30 G1 X125 F50 – подача 50мм/мин

N35 Y45 FB=80 – подача 80мм/мин

N40 X-125 – подача 50мм/мин

С помощью G4 можно прервать обработку детали между двумя кадрами УП на запрограммированное время, задаётся отдельным кадром. Формат:

G4 F… - время в секундах или G4 S… - задержка времени в оборотах шпинделя. Примеры: N25 G4 F3 – пауза 3с, N50 G4 S30 – ожидать 30 оборотов шпинделя, если для шпинделя задано 300мин$^{-1}$ и процентовка числа оборотов 100%, то время ожидания будет 6с (30/300=0,1мин).

4. **Программирование перемещений** G0, G1

G0 – используется для быстрого позиционирования инструмента, для обхода детали или для подвода к точкам смены инструмента. Действует до отмены.

G1 – линейная интерполяция с заданной подачей. Действует до отмены.

Примеры: N20 G0 X20 Y20 Z2 – переход к стартовой позиции

N30 G1 Z-2 F40 – линейная интерполяция с заданной подачей

5. **Управление траекторией** G9, G64

6. **Функция точного останова G9, используется тогда, когда необходимо острых внешних углов или чистовая обработка внутренних углов по размеру с максимальной точностью.**

В режиме управления траекторией G64 контур изготавливается с постоянной скоростью движения по траектории (нет торможений на границах кадров). Равномерная скорость способствует лучшим условиям резания, улучшает качество поверхностей и уменьшает время обработки. В режиме управления траекторией не осуществляется точного подвода к запрограммированным переходам контура. Острые углы создаются с помощью G60 или G9. В контурном режиме G64 инструмент перемещается с максимально возможной постоянной путевой скоростью (нет торможений на границах кадров). При изменении направления движения, переходы между элементами контура сглаживаются, то есть заданные позиции отрабатываются не так точно как запрограммированно и таким образом углы обрабатываются непрерывно. При этом скорость значительно снижается из-за учитывания границ ускорений и фактора перегрузки осей. Погрешность обработки увеличивается при возрастании скорости. Значительное преимущество обработки в контурном режиме заключается в том, что станочные оси перемещаются очень плавно и при этом можно достичь очень высокого качества поверхности. Пример: N1 G0 G09 G90 X… Y… - точный останов в заданном положении

## **Круговая, винтовая интерполяция** G2, G3

**Круговая интерполяция** предназначена для обработки полных окружностей или их сегментов и осуществляется при помощи команд G2, G3 (движение по часовой стрелке или против часовой стрелки) с указанием рабочей плоскости (G17-G19). Можно создавать окружности вне выбранной рабочей плоскости (не при указании угла и спиральной линии). В этом случае плоскость определяют адреса осей, которые указываются в качестве конечной точки окружности. Способы задания кругового движения:

1. Программирование через центр и конечную точку. Формат:

G2/G3 X Y Z I=AC(…) K=AC(…) – центр и конечная точка заданы абсолютно, относительно нулевой точки детали, или

G2/G3 X Y Z I J K – центр задан относительно начальной точки окружности. Если при программировании окружности через центр не задаётся конечная точка, то в этом случае будет обработана полная окружность. Параметры интерполяции I, J, K со значениями 0 могут быть опущены, например: вместо N10 G2 X10 Y50 I-50 J0 можно записать N10 G2 X10 Y50 I-50

2. Программирование через радиус и конечную точку. Формат:

G2/G3 X Y Z CR= - радиус окружности CR=… и конечная точка окружности. Если угол окружности меньше или равен $180^0$, то CR=+…, если больше $180^0$, то CR=-…. Полная окружность $360^0$ не может быть запрограммирована при помощи радиуса CR=…, поэтому необходимо воспользоваться способом задания через конечную точку и центр.

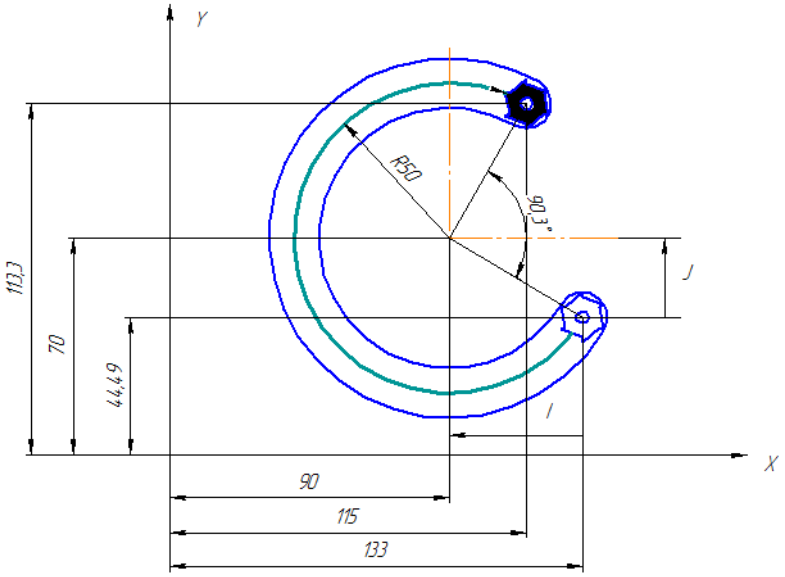
3. Программирование через конечную точку и угол. Формат:
G2/G3 X… Y… Z… AR= - угол AR=… конечная точка окружности. Значение углов от $0^0$ до $360^0$. Полная окружность $360^0$ не может быть запрограммирована при помощи угла AR=…, поэтому необходимо воспользоваться способом задания через конечную точку и центр.

4. Программирование через точку и угол. Формат:
G2/G3 I J K AR=… (угол AR=… центр под адресами I, J, K). Значение углов от $0^0$ до $360^0$. Полная окружность $360^0$ не может быть запрограммирована при помощи угла AR=…, поэтому необходимо воспользоваться способом задания через конечную точку и центр.

5. Программирование через полярные координаты. Формат:
G2/G3 AP=…, где AP=±… - полярный угол (положительное значение указывает направление отсчёта против часовой стрелки), RP=… - полярный радиус окружности. Полюс должен лежать в центру окружности.

6. Программирование через промежуточную и конечную точку. Формат: CIP X Y Z I1=AC(…) J1=AC(…) K1=AC(…), где c=CIP (Circle with intermediate point) – круговая интерполяция через промежуточную точку (действует модально), I1=J1=K1= - координаты промежуточной точки

Пример программирования:




N10 G0 G90 X133 Y44.48 S800 M3
N20 G17 G1 Z-5 F1000
N30 G2 X115 Y113.3 I-43 J25.52
или
N30 G2 X115 Y 113.3 I=AC(90) J=AC(70)
или
N30 G2 X115 Y 113.3 CR=-50
или
N30 G2 AR=269.31 I-43 J25.52
или
N30 G2 AR=269.31 X115 Y113.3

или
N29 G111 X90 Y70
или
N30 G2 RP=50 AP=-269.3
или
N30 CIP X115 Y113.3 I1=IC(-85.35) J1=IC(-35.35)

**Фаска**, **закругление** (CHF, CHR, RND, RNDM)

В угол контура могут быть вставлены следующие элементы: фаска или закругление. Программирование:

CHF=**…** - снятие фаски угла контура (по длине фаски)

CHR=**…** - снятие фаски угла контура (по ширине фаски, например, $2x45^0$)

Фаска обрабатывается сразу же в том кадре, в котором она запрограммирована.

RND=**…** - закругление угла контура, действует на кадр

RND**M**=**…** - модальное закрепление (одинаковое закругление нескольких последовательных углов контура). При RNDM=0 закругление отменяется. Если запрограммированные значение фаски или сопряжение слишком велики для элементов контура, то значения фаски и сопряжения автоматически уменьшается. Примеры программирования:

N10 G17 G94 G0 X0 Y0 F100
N20 G1 X10 CHR=2 – фаска $2x45^0$
N30 Y10 CHF=4 – фаска (длина 4мм)
N50 RNDM=2 – модальное закругление (R2)
N60 Y20
N70 X30
N100 Y40

**Коррекция радиуса инструмента** G40, G41, G42

**Обход углов** G450, G451, DISC=**…**

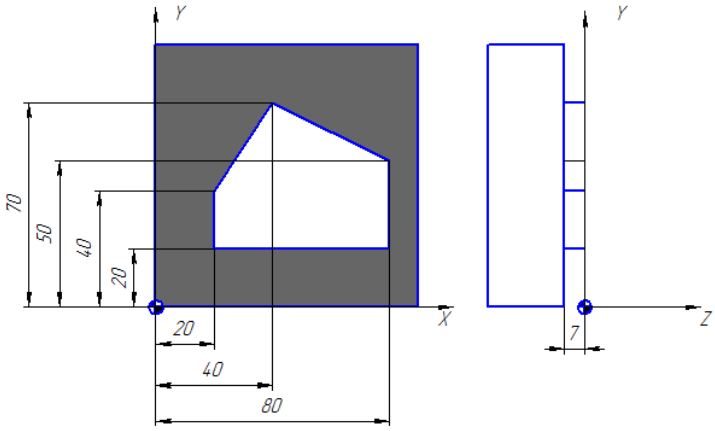
1. При создании программы во внимание не принимаются размеры инструмента, программируются только перемещения инструмента согласно геометрии, определенной на чертеже. При изготовлении детали перемещением инструмента надо управлять так, чтобы установленным инструментом можно было бы обрабатывать заданный контур. При включенной коррекции радиуса инструмента СЧПУ автоматически вычисляет для различных инструментов соответствующие эквидистантные пути перемещения инструмента. Программирование:

G41 – включение коррекции радиуса инструмента слева от контура
G42 – включение коррекции радиуса инструмента справа от контура
G40 – выключение коррекции радиуса инструмента
Пример:
N12 T1 D1 M6
N13 G42 G0 G90 X Y Z – включение коррекции радиуса инструмента

N14 G1 X Y Z – перемещение с коррекцией радиуса инструмента

Для вычисления траекторий инструмента СЧПУ необходима следующая информация: номер инструмента T, номер кромки D, направление обработки G41(G42), рабочая плоскость G17-G19. В кадре с G40, G41 или G42 должна быть запрограммирована команда движения с G0 или G1. В этой команде движения должна быть указана как минимум одна ось выбранной рабочей плоскости. Если при включении указывается только одна ось, то последняя позиция второй оси автоматически дополняется и перемещение осуществляется в обеих осях. Смена рабочей плоскости G17, G18, G19 при включенной G41/G42 невозможна. Смена направления коррекции G41/G42, G42/G41 может программироваться без промежуточного включения G40.

Пример фрезерования контура:



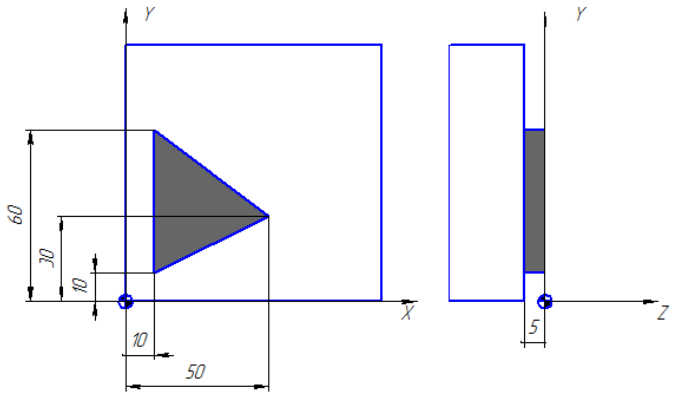

N10 G0 Z100
N20 G17 T1 M6 – смена инструмента
N30 G0 X0 Y0 Z1 S300 M3 D1
N40 G1 Z-7 F50
N50 G41 X20 Y20 – включение коррекции слева
N60 Y40 – фрезерование контура
N70 X40 Y70
N80 X80 Y50
N90 Y20
N100 X20
N110 G40 G0 Z100 M5 – отмена коррекции
N120 M30

2. С помощью G450/G451 производится обход внешних углов.

G450 – инструмент обходит углы детали по круговой траектории с радиусом инструмента

G451 – точка пересечения, инструмент осуществляет свободное резание в углу детали

В этом примере на всех наружных углах вставляется переходный радиус (кадр N30). Благодаря этому удаётся избежать остановки и свободного хода инструмента для смены направления.




N10 G17 G0 X35 Y0 Z0 F50
N20 G1 Z-5
N30 G41 G450 X10 Y10 – включение режима коррекции
N40 Y60 – фрезерование контура
N50 X50 Y30
N60 X10 Y10
N70 G40 X-20 Y50 – выключение коррекции

## Подвод к контуру и отвод от контура G147, G148, G247, G248, G347, G348

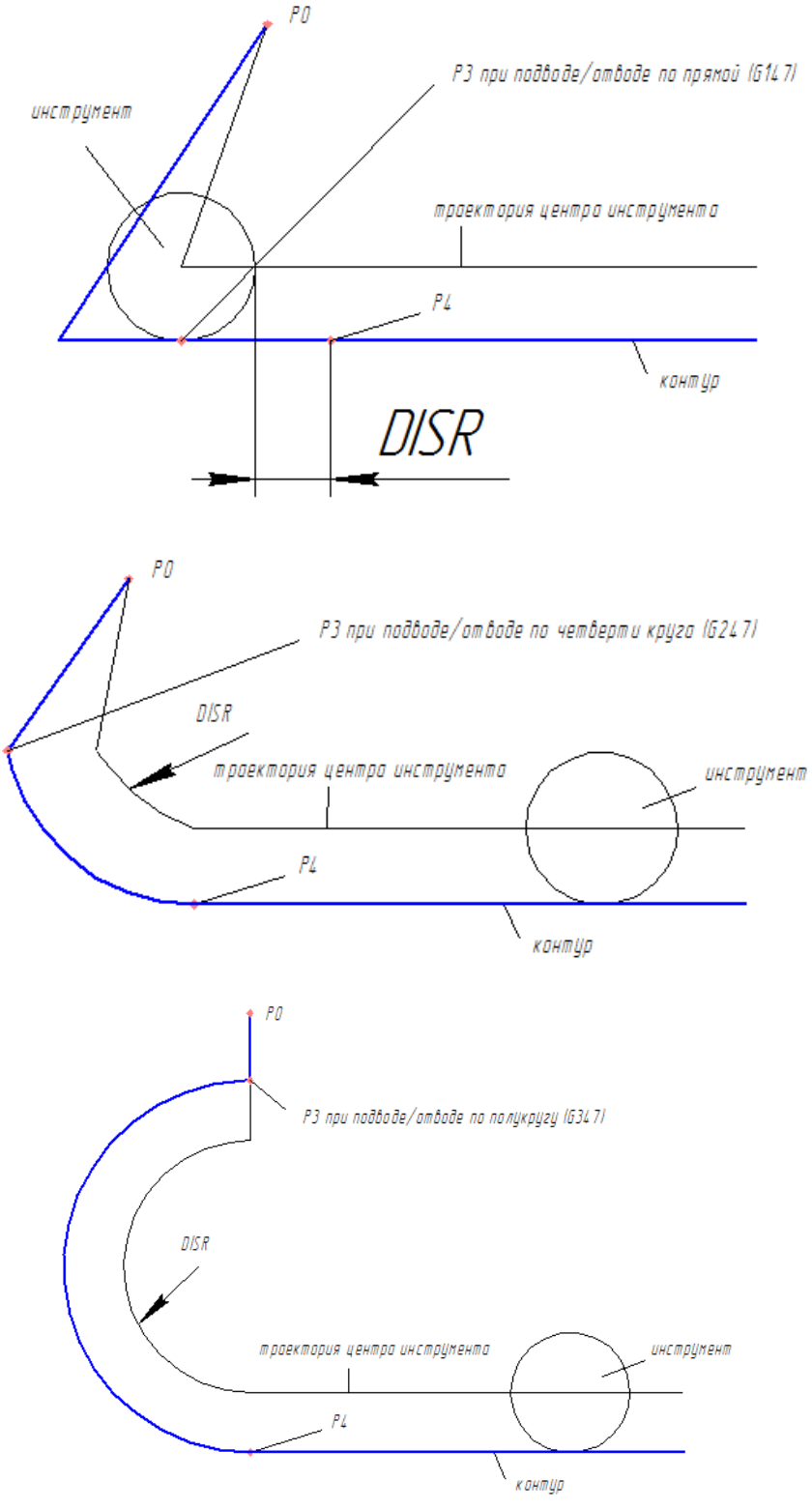
Функция мягкого подвода и отвода служит для того, чтобы осуществить подвод по касательной в стартовой точке контура, независимо от положения исходной точки. Эта функция используется преимущественно вместе с коррекцией радиуса инструмента, но это не является обязательным условием. Движения подвода и отвода состоят максимально из 4-х вспомогательных движений:

- стартовая точка движения P0
- промежуточные точки P1 P2 P3
- конечная точка P4

Точки P0 P3 P4 всегда определены. Промежуточные точки P1 и P2 могут отсутствовать в зависимости от параметрирования и геометрических соотношений. Программирование:

1. **Выбор контура подвода или отвода**

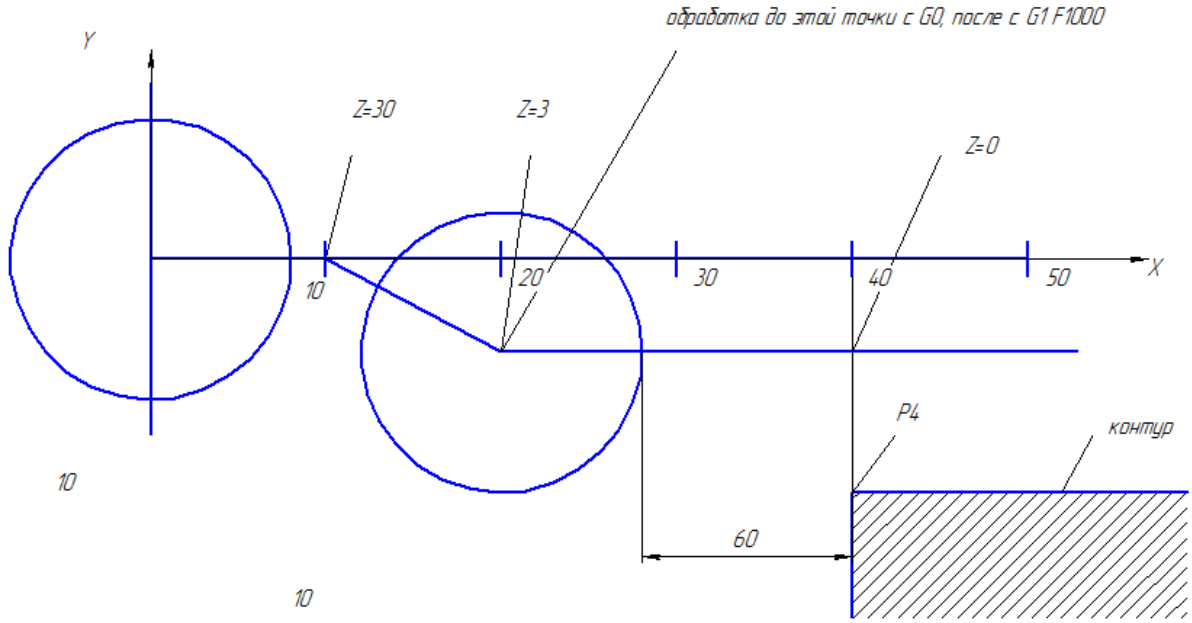
G147 – подвод по прямой
G148 – отвод по прямой
G247 – подвод по четверти круга
G248 – отвод по четверти круга
G347 – подвод по полуокружности
G348 – отвод по полуокружности




Движения подвода и отвода представлены с промежуточной точкой ft (при одновременной активации коррекции радиуса инструмента)
DISR=... - расстояние от кромки фрезы до НТ контура (G147/G148) или R траектории це1гтра инструмента при G247/G248, G347/G348

**Пример:**

Обработка контура с подводом по прямой (G147), расстояние от кромки фрезы до НТ контура (Р4) 13мм, подвод по оси Z на ускоренном ходу с недобегом до плоскости обработки 3мм (DISCL=3).

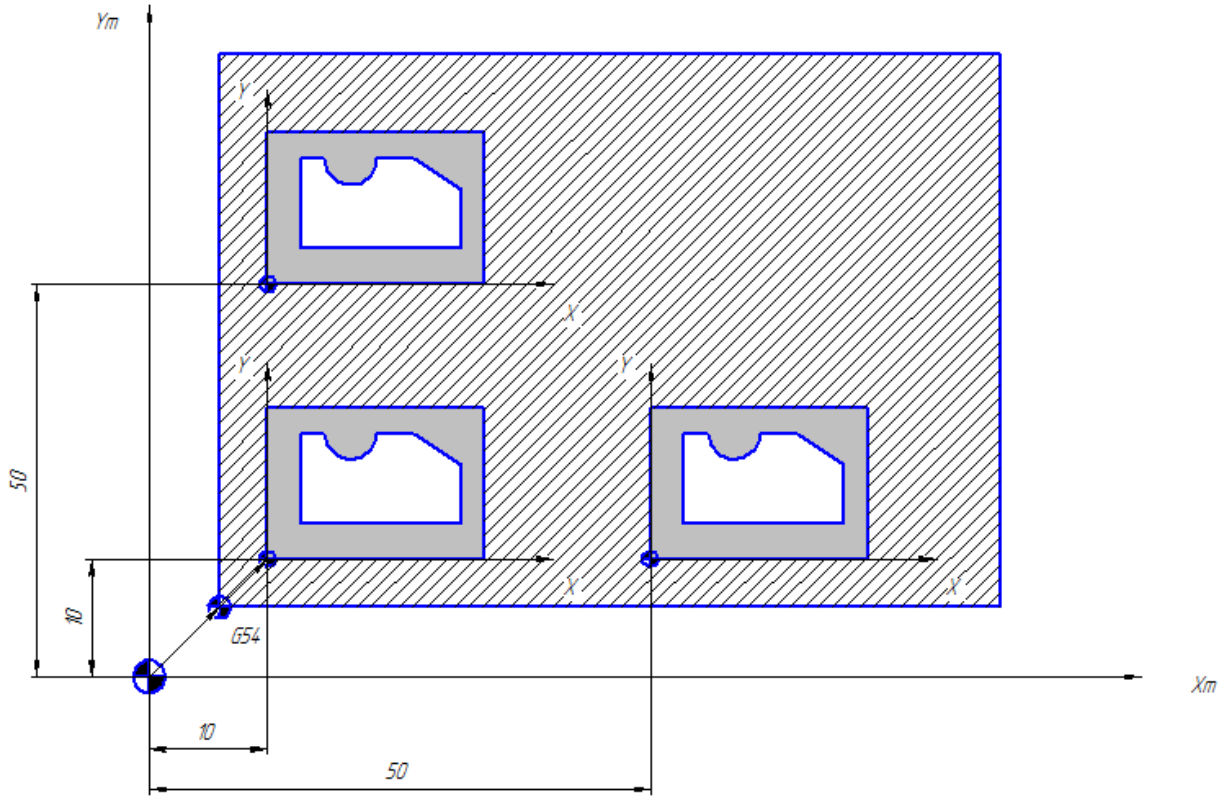



N10 G90 G54 G0 X0 Y0 Z30 D1 T1
N20 X10
N30 G41 G147 DISCL=3 DISR=13 Z0 F1000 – перемещение в т. P3 на БХ
N40 G1 X40 Y-1
N50 G1 X50
Кадры N30/N40 могут быть заменены на один кадр:
N30 G41 G147 DISCL=3 DISR=13 X40 Y-10 Z0 F1000
или
N30 G41 G147 DISCL=3 DISR=13 F1000
N40 G1 X40 Y-10 Z0

**Программируемое смещение нулевой точки TRANS, ATRANS** (программирование смещения нулевой точки в направлении указанной оси). Благодаря этому можно работать с изменяемыми нулевыми точками, например, при повторяющихся ходах обработки на различных позициях детали. Формат оператора имеет сл. вид:

TRANS X Y Z - абсолютное смещение нулевой точки относительно G54 - G599. ATRANS - смещение относительно предыдущей нулевой точки. Команда TRANS (без значений) сбрасывает все предыдущие фреймы.

Пример фрезерования:



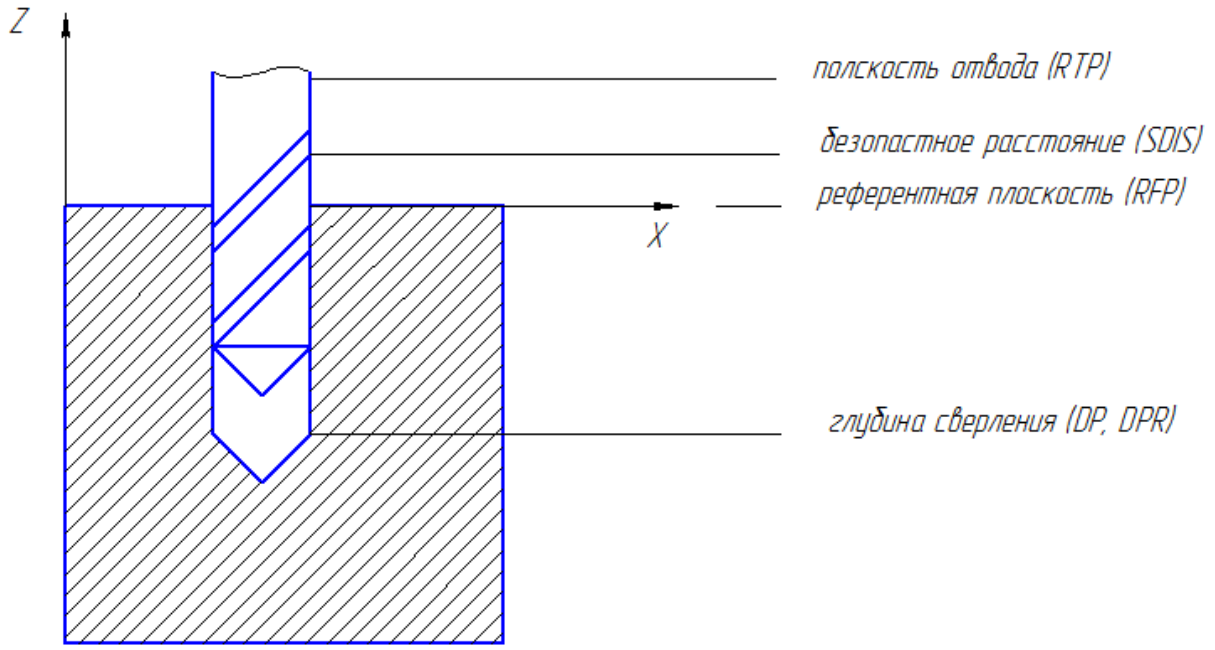

N10 G1 G54… – нулевая точка детали
N20 G0 X0 Y0 Z2 – подвод к точке старта
N30 TRANS X10 Y10 – абсолютное смещение XY
N40 L10 – подпрограмма обработки
N50 TRANS X50 Y10 – абсолютное смещение XY
N60 L10 – подпрограмма обработки
N70 ATRANS X-40 Y40 – относительное смещение XY
N80 L10 – подпрограмма обработки
N90 TRANS – отмена смещения нуля

**Сверлильные, резьбовые циклы CYCLE 81** - CYCLE 84

Циклы являются технологическими подпрограммами, с помощью которых можно произвести соответствующий процесс обработки универсальным, например, нарезание резьбы или фрезерование выемки.

Сверлильные циклы имеют два вида параметров:

- геометрические
- параметры обработки
    - Геометрические параметры идентичны для всех циклов сверления, сверления по схеме и циклов фрезерования. Они определяют референтную или базовую плоскость (плоскость относительно нулевой точки детали по оси Z) плоскость отвода, безопасное расстояние, а так же абсолютную и относительную глубину сверления. Геометрические параметры описываются однократно при первом цикле CYCLE81.




- Параметры обработки имеют для различных циклов различное значение и действие. Поэтому они описываются в каждом цикле отдельно. Скорость вращения шпинделя, направление вращения, подача и позиция сверления (координаты отверстия, подвод инструмента, СОЖ) задаются перед циклом.

Циклы сверления могут действовать модально (до отмены) или не модально, т.е. после перемещения инструмента в следующую позицию сверления необходимо вновь вызывать цикл.

## CYCLE **81 (сверление, центрование)**

Формат цикла имеет сл. вид:

CYCLE 81 (RTP, RFP, SDIS, DP, DPR), где:

- RTP (Re Traction Plane) – плоскость отвода (абсолютное значение)
- RFP (Re Ference Plane) – референтная плоскость (абс.)
- SDIS (Safety Distance) – безопасное расстояние (задаётся без знака)
- DP (De Pth) – конечная глубина сверления (абс.)
- DPR (De Pth Relative) - конечная глубина сверления относительно референтной плоскости (задаётся без знака)

Плоскости RFP и RTP должны иметь различные значения, т.е. плоскость отвода задаётся перед референтной плоскостью. Если значения одинаковы (RFP = RTP), то относительное задание глубины сверления не допускается, появляется сбойное сообщение и цикл не отрабатывается. Не отрабатывается цикл и в том случае, когда расстояние от плоскости отвода до глубины сверления меньше, чем от референтной плоскости. Безопасное расстояние (SDIS) задаётся относительно референтной плоскости и направление, в котором действует SDIS, определяется в цикле автоматически.

Пример: произвести сверление трёх отверстий по циклу CYCLE 81 с различными параметрами цикла.

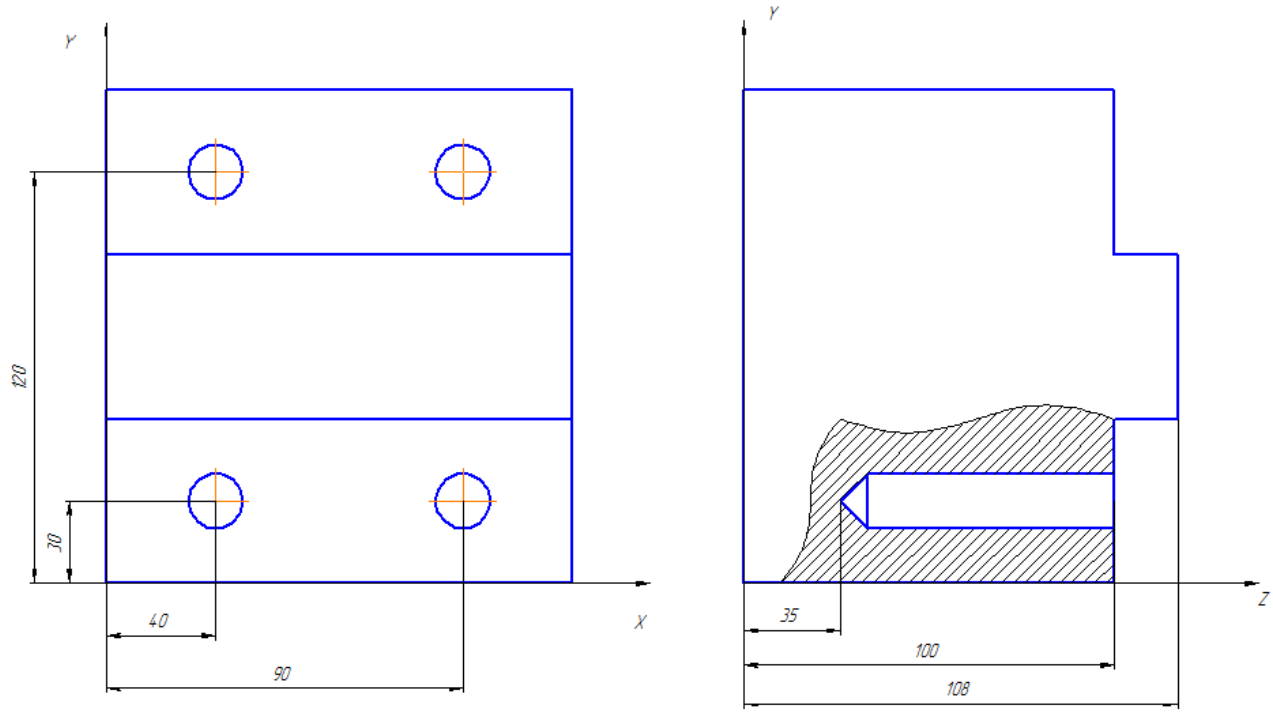



Рис. 1

Управляющая программа (см. рис. 1)
N1 G90 G54 T1 D1 M6 – задание нулевой точки детали, номера инструмента
N2 S600 M3 F50 – задание технологических значений
N3 G0 X40 Y120 Z110 – выход на первую позицию сверления
N4 MCALL CYCLE 81 (110, 100, 2, 35) – модальный вызов цикла (MCALL – ключевое слово) с абсолютной глубиной сверления
N5 X40 Y120 – сверление первого отверстия (координаты повторить в циклы)
N6 Y30 – сверление второго отверстия
N7 X90 – сверление третьего отверстия
N8 MCALL – отмена цикла
N9 M9 – выключение СОЖ
N10 M5 – выключение вращения шпинделя
N11 L60 – смена паллет
N12 M30 – конец программы
*Примечание:*

- если не задавать MCALL, то после перемещения га координаты сл. отверстия необходимо опять вызывать цикл и т.д.
- дробная часть отделяется от целого числа через точку.

## CYCLE 82 (сверление, цекование)

Цикл аналогичен циклу CYCLE 81, только с выдержкой времени на глубине сверления. Формат цикла имеет сл. вид:
CYCLE 82 (RTP, RFP, SDIS, DP, DPR, DTB)
Первые пять параметров такие же, как и для цикла CYCLE 81

- DTB (Dwell Time at Bottom) - выдержка времени в секундах

## CYCLE 83 (глубокое сверление)

Формат цикла имеет сл. вид:

CYCLE 83 (RTP, RFP, SDIS, DP, DPR, FDEP, FDPR, DAM, DTB, DTS, FRF, VARI, AXN*, MDEP*, VRT*, DTD*, DIS1*), где

- RTP, RFP, SDIS, DP, DPR - назначение параметров как для CYCLE 81
- FDEP (First DEPth) - первая глубина сверления (абс.)
- FDPR (First DEPth Relative) - первая глубина сверления относительно
- референтной плоскости (задаётся без знака)
- DAM (Degression Amount) - дегрессионная величина, определяет величину сл. хода сверла (уменьшение хода). Если DAM > 0, то второй цикл сверления уменьшается на величину DAM (FDEP- DAM) при условии что ход сверления > DAM. Если второй ход < DAM, то он выполняется за один этап. Последующие шаги сверления будут соответствовать значению дегрессии до тех пор, пока остаточная глубина сверления больше, чем двойное значение дегрессии. Последние два хода сверления делятся и проходятся равномерно. Если DAM =0, то нет дегрессии, все шаги сверления производятся на величину FDEP.
- DTB (Dwell Time at Bottom) - выдержка времени на конечной глубине сверления (облом стружки)
- DTS (Dwell Time at in feed Start) - пауза в начальной точке (SDIS) до удаления стружки, выполняется только при VARI = 1 (тин обработки)
- FRF (Feed Reduction Factor) - коэффициент понижения для действующей подачи, начиная со второго хода (задаётся без знака), Диапазон значений: 0.001 ...1
- VARI (VARIant) - тип обработки. VARI - 0 - облом стружки (возврат в заданную величину параметра VRT, если VRT4) или VRT опущен, то возврат на 1мм), VARI = 1 - удаление стружки (возврат на ил. SDIS)
- AXN (AXis)- выбор оси инструмента. Значения: AXN 1,2, 3 (X, Y, Z)

| Command | Plane | Vertical infeed axe |
|---|---|---|
| G17 | X/Y | Z |
| G18 | Z/X | Y |
| G19 | Y/X | X |

| | G17 | G18 | G19 |
|---|---|---|---|
| X | AXN=1 | AXN=2 | AXN=3 |
| Y | AXN=2 | AXN=3 | AXN=1 |
| Z | AXN=3 | AXN=1 | AXN=2 |

- MDEP (Min. drilling DEPth) - мин. глубина сверления, назначается при вычислениях хода сверления через коэффиц. дегрессии (см. DAM < 0)
- VRT (Variable ReTurn path) - величина отвода при обломе стружки (при VARI=0). Значения: VRT> 0 - отвод на указанную величину, VRT=0 - отвод на 1мм

- DTD (Dwell Time) пауза на конечной глубине сверлении
- DIS1 (Distance) - недобег при возврате инструмента на сл. шаг сверления (при VARI = 1). Значения: DIS1 > 0 - недобег на указанное значение, DIS1 =0 - автоматическое вычисление.

Параметры, помеченные символом * можно опускать.

Пример: произвести глубокое сверление для первого отверстия с нулевой выдержкой времени (DTS), тип обработки с обломом стружки (VARI=0), конечная глубина сверления и шаг сверления задаётся в абсолютной системе. Для второго отверстия выдержка времени 1с, тип обработки с удалением стружки (VARI=1), конечная глубина сверления задаётся относительно референтной плоскости.

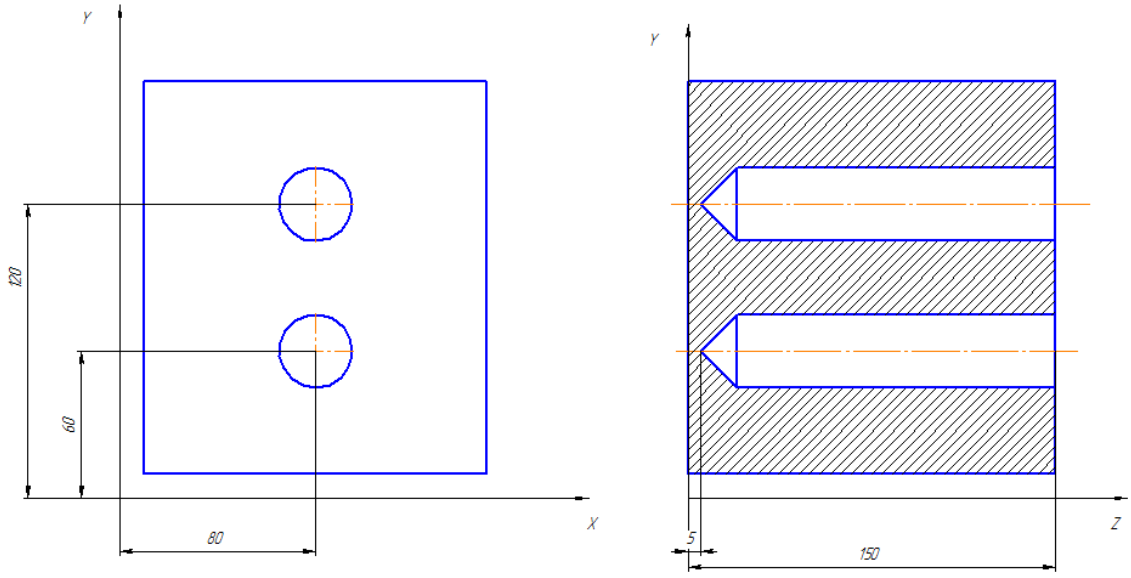



Рис. 3

*Примечание*: недобег инструмента при возврате в точку прерывания цикла составляет 0,6мм при глубине сверления до 30мм, в остальных случаях глубина сверления делится на 50 (при этом значение ограничено до 7мм)

Фрагмент УП будет иметь сл. вид (см. рис.3)

…..

DEF REAL RTP=155, RFP=150, SDIS=2, - определение параметров

DP=5, DPR=145, FDEP=100, FDPR=50, - присвоение значений

DAM=20, DTS=3, DTB=0, FRF=1, VARI=0 - присвоение значений

N10 G17 G90 G0 G55 F50 S600 M3 – задание технологических значений

N15 T3 D1 Z155 – выход на плоскость отвода

N20 X80 Y120 – выход в первую позицию сверления

N25 CYCLE 83 (RTP, RFP, SDIS, DP, , FDEP, , DAM, , , FRF, VARI) – вызов цикла, параметр глубины в абсолютной системе

N30 Y60 – выход на сл. позицию сверления

N35 FRF=0.5, VARI=1 – присвоение значений

N40 CYCLE 83 (RTP, RFP, SDIS, , DPR, , FDPR, DAM, , DTS, FRF, VARI) – вызов цикла с относительным заданием конечной глубины и 1-ой глубины сверления, коэффициент подачи 0.5

N45 M30 – конец программы

**CYCLE 84 (нарезание резьбы метчиком без компенсирующего патрона)**

Формат цикла имеет сл. вид:

CYCLE 84 (RTP, RFP, SDIS, DP, DPR, DTB, SDAC, MPIT, PIT, POSS, SST, SST1, AXN*, PTAB*, TECH*, VARI*, DAM*, VRT* ), где

Параметры, помеченные символом * можно опускать

- RTP, RFP, SDIS, DP, DPR - назначение параметров как для CYCLE 81
- DTB - выдержка времени в секундах на глубине резьбы (облом). Для глухих отверстий применять не рекомендуется.
- SDAC (Spindle Direction After Cycle) - направление вращения по окончании цикла. Значения: 3, 4 или 5 (M3, M4, M5)
- MPIT (Metrical PITch) - размер метрической резьбы (со знаком). Диапазон значений: 3 (для M3) .... 48 (для M48), знак определяет направление вращения в резьбе: +3 В правая резьба, -3 - левая резьба
- PIT (PITch) - шаг резьбы (со знаком). Диапазон значений: 0,001 ... 2000.000 мм. Знак определяет направлению вращения в резьбе.
- TOSS (POSition of Spindle) - ориентированный останов шпинделя перед нарезанием резьбы (в градусах)
- SST (Spindle Speed for Tapping) - число оборотов шпинделя при нарезании резьбы
- SST1 (Spindle Speed for Tapping) - число оборотов шпинделя при отводе, если 0, то возврат происходит на той же скорости вращения, которая записана в параметре SST
- AXN - выбор оси инструмента
- PTAB - единица измерения шага резьбы. Значения:
  0- шаг резьбы в соответствии с запрограммированной системой единиц (дюйм овая/м етрическая)
  1- шаг резьбы в мм
  2- шаг резьбы в витках резьбы на дюйм
  3- шаг резьбы в дюймах на оборот
  TECH - технологические настройки.
- VARI режим обработки. Значения:
  0- нарезание внутренней резьбы за одно движение
  1- глубокое нарезание внутренней резьбы с ломкой стружки
  2- глубокое нарезание внутренней резьбы с удалением стружки
- DAM определяет величину следующего хода метчика (см. CYCLE 83).
- VRT величина отвода при обломе стружки (при VARI = I). Значения: VRT> 0 - отвод на указанную величину, VRT » 0 - отвод на 1мм Параметры, помеченные символом * можно опускать.

*Примечание*: направление вращения при нарезании резьбы в цикле меняется всегда автоматически Резьба может задаваться по выбору либо через размер, либо через шаг резьбы. Не используемый параметр пропускается при вызове цикла или ему присваивается значение ноль.

Пример: произвести нарезание резьбы М5 с нулевой выдержкой времени, задание глубины производится в относительной системе.

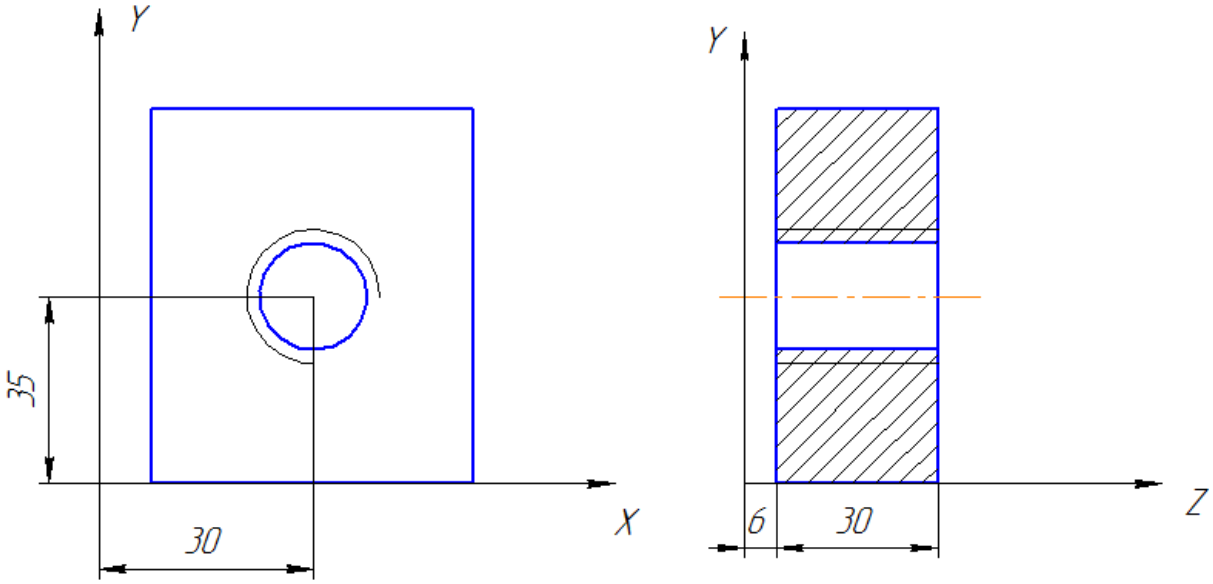



Рис. 4

Формат УП будет иметь сл. вид (см. рис.4)

…..

N15 G90 G54 T4 D1 M6 – задание технологических значений
N20 G0 X30 Y35 Z50 – выход в позицию нарезания резьбы
N25 CYCLE 84 (50, 36, 2, , 30, , 3, 5, , 90, 200, 500) – вызов цикла
N30 G0 Z50 Y50 M5 – отвод, выключение вращения шпинделя
N35 M30 – конец программы

**CYCLE 840 (нарезание резьбы с компенсирующим патроном)**

Инструмент нарезает резьбу (без датчика или с датчиком) с запрограммированной скоростью вращения шпинделя и скоростью подачи до заданной глубины резьбы. Формат цикла имеет сл. вид:

CYCLE 840 (RTP, RFP, SDIS, DP, DPR, DTB, SDR, SDAC, ENC, MPIT, PIT, AXN*, PTAB*, TECH*), где

- RTP, RFP, SDIS, DP, DPR, - назначение параметров как для CYCLE 81
- DTB - пауза в секундах на глубине резьбы, действует в зависимости от выбора технологического варианта в параметре ENC
- SDR (Spindle Direction for Retraction) - направление вращения для отвода. Значения: 0 (с датчиком, автоматическая смена направления вращения), 3 или 4 (как для М3 или М4).
- SDAC - направление вращения по окончании цикла. Значения: 3, 4 или 5 (как для М3, М4 или М5), так как цикл может выполняться и модально, то для выполнения сл. нарезаний резьбы ему необходимо направление вращения. Оно программируется в параметре SDAC и соответствует направлению вращения, записанному перед первым вызовом цикла в вышестоящей программе. Если SDR=0, то значение параметра SDAC не

имеет значения в цикле, оно может быть опущено при параметрировании.

- ENC (ENCoder)- нарезание резьбы /без датчика. Значения:

0- с датчиком, без паузы

1- без датчика, подача запрограммирована перед циклом

Если нарезание резьбы должно производиться без датчика, несмотря на то, что датчик имеется, параметру ENC присваивается значение 1. Если же датчика нет и значение параметра равно нулю, то он не учитывается в цикле.

- MPIT - размер метрической резьбы (шаг определяется автоматически). Диапазон значений: 3 (для M3)... 48 (для M48)
- PIT - шаг резьбы. Значения: 0,001 ... 2000 мм. Если PTAB=0 или 1,то шаг в мм. Если PTAB=2, то шаг в витках на дюйм.

Параметр для шага имеет значение только для нарезания резьбы с датчиком. Исходя из числа оборотов шпинделя и шага резьбы, цикл вычисляет значение подачи. Резьба может задаваться по выбору либо через размер (только для метрической резьбы в диапазоне от M3 до M48), либо через шаг резьбы. Не используемый параметр пропускается при вызове цикла или ему присваивается значение ноль.

- AXN - выбор оси инструмента
- PTAB - единица измерения шага резьбы (см. CYCLE 84)
- TECH-технологические настройки

Параметры, помеченные символом * можно опускать

*Примечание:*

- перед вызовом цикла программируется .направление вращения шпинделя с M3 или M4
- во время нарезания резьбы переключатель коррекции скорости подачи и вращения шпинделя должны быть зафиксированы назначении 100%

Пример1 (резьба без датчика). Параметры направления вращения SDR и SDAC должны быть заданы предварительно, параметру ENC присваивается значение 1, глубина задаётся абсолютно. Параметр шага резьбы PIT пропускается. Используется компенсирующий патрон.

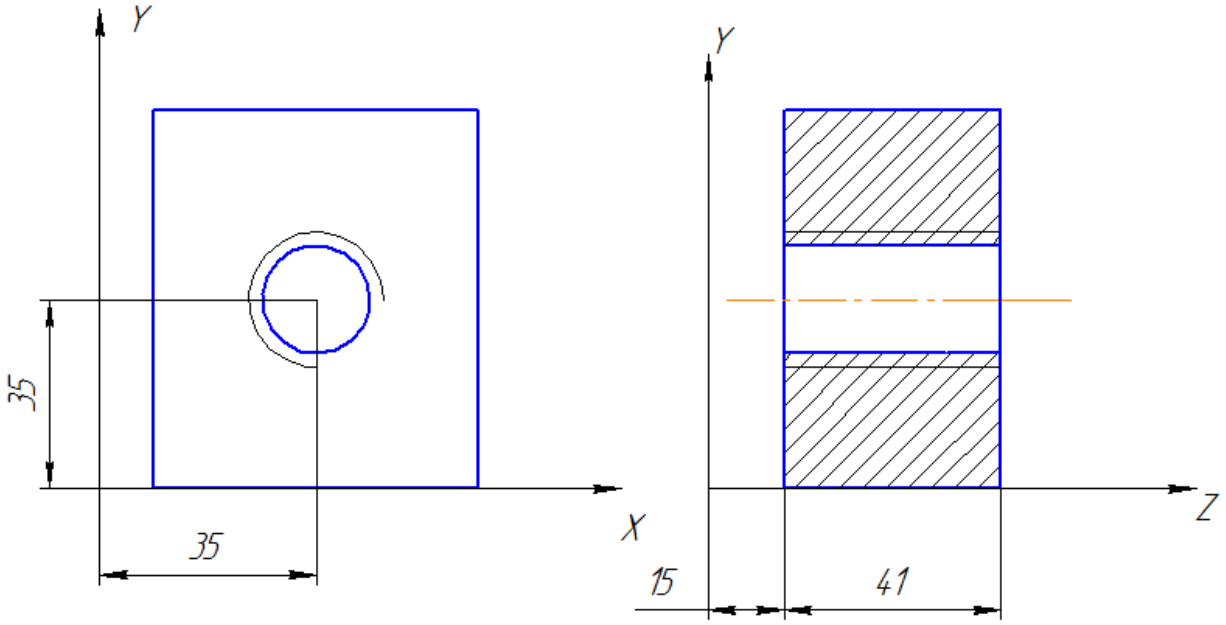



Рис. 5

Фрагмент УП будет иметь сл. вид (см. рис.5)

…..

N10 G90 G0 D1 T2 S100 M3 – задание технологических значений

N15 G17 X35 Y35 Z60 – выход в позицию нарезания резьбы

N20 G1 F200 – определение рабочей подачи

N25 CYCLE 840 (60, 56, 15, , 1, 4, 3, 1) – вызов цикла

N30 G0 Z100 Y100 M5 – отвод, выключение вращения шпинделя

N35 M30 – конец программы

**Расточные циклы CYCLE 85** - CYCLE 89

Формат цикла имеет сл. вид:

CYCLE 85 (RTP, RFP, SDIS, DP, DPR, DTB, FFR, RFF), где

- RTP, RFP, SDIS, DP, DPR, - назначение параметров как для CYCLE 81
- DTB - выдержка времени в секундах на конечной глубине (облом стружки)
- FFR (Forward Feed Rate)-рабочая подача
- RFF (Retraction Feed) - подача при возврате

Пример: произвести растачивание отверстия с нулевой выдержкой времени, задание глубины производится в относительной системе.

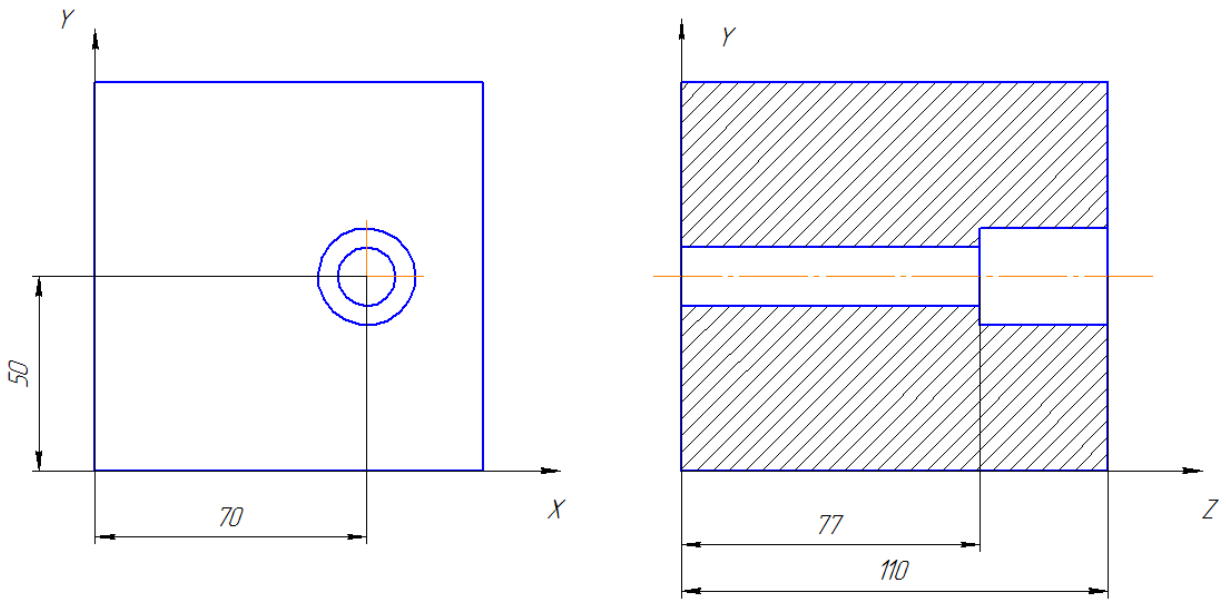



Рис. 6

Фрагмент УП будет иметь сл. вид (см. рис.6)

…..

N10 G90 G0 G54 X70 Y70 S300 M3 – задание технологических значений

N15 T5 D1 Z115 M8 – выход в позицию нарезания резьбы

N20 CYCLE 85 (115, 110, 4, , 33, , 60, 90) – вызов цикла

N25 M30 – конец программы

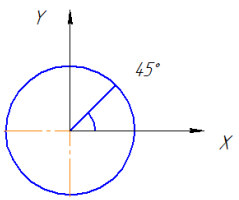
**Расточка 2** - CYCLE 86

Формат цикла имеет сл. вид:

CYCLE 86 (RTP, RFP, SDIS, DP, DPR, DTB, SDIR, RPA, RPO, RPAP, POSS)

- RTP, RFP, SDIS, DP, DPR, DTB - параметры как для CYCLE 82
- SDIR (Spindle DIRection) направление вращения шпинделя. Значения: 3 (как M3), 4 (как M4), при другом значении появляется ошибка 61102
- RPA (Retraction Position Abscissa) 1 отвод на абсциссу по достижении конечной глубины и ориентированного останова (задаётся со знаком)
- RPO (Retraction Position Ordinate) - отвод на ординату по достижении конечной г лубины и ориентированного останова (задаётся со знаком)
- RPAP (Retraction Position APplicate) - отвод на аппликату по достижении конечной глубины и ориентированного останова (задаётся со знаком)
- POSS (POSition of Spindle) - позиция ориентированного останова шпинделя по достижении конечной глубины (в градусах)

*Примечание*: цикл CYCLE 86 может использоваться тогда, когда шпиндель, имеет возможность перейти в режим управления положением шпинделя.



Пример: произвести растачивание отверстия с выдержкой времени 2с, конечная глубина программируется абсолютно, безопасное расстояние не задаётся, шпиндель ориентируется в поз. 45°.
Фрагмент УП будет иметь сл. вид (см. рис.6)
…..
N10 G90 G0 G54 X70 Y50 S300 M3 – задание технологических значений
N15 T5 D1 Z115 M8 – выход в позицию нарезания резьбы
N20 CYCLE 85 (115, 112, ,77 , , 2, 3, -1, -1, +1, 45) – вызов цикла
N25 M30 – конец программы

**Расточка 3** — CYCLE 86
При расточке после достижения заданной глубины происходит неориентированный останов шпинделя (M5), программируемый останов (MO), затем возврат на плоскость отвода с помощью кн. NC - START с G0
Цикл выполняет сл. движения:

- выход на безопасное расстояние на ускоренном ходу с G0
- движение на глубину расточки с подачей, заданной перед циклом
- останов шпинделя (M5)
- нажатие кнопки NC - STAR’S
- возврат на плоскость отвода с G0

Формат цикла имеет сл. вид:
CYCLE 87 (RTP, RFP, SDIS, DP, DPR, SDIR)
Назначение параметров такое же, как для цикла CYCLE 86
**Расточка 4** - CYCLE 88
Цикл аналогичен циклу CYCLE 87, только с выдержкой времени на конечной глубине расточки.
Формат цикла имеет сл. вид:
CYCLE 88 (RTP, RFP, SDIS, DP, DPR, DTB, SDIR)
**Расточка 5**-CYCLE 89
При расточке производится движение на конечную глубину на рабочей подаче, выдержка времени, возврат на безопасное расстояние на рабочей подаче и перемещение на плоскость отвода с G0.
Формат цикла имеет сл. вид:
CYCLE 89 (RTP, RFP, SDIS, DP, DPR, DTB)
Назначение параметров такое же, как для цикла CYCLE 82

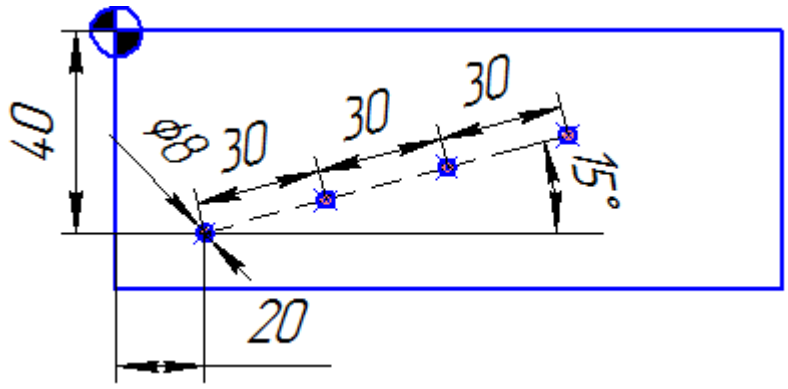
## Циклы формирования отверстий

Циклы формирования отверстий описывают только геометрию расположения отверстий в плоскости. Связь с циклом сверления создаётся через модальный вызов этого цикла сверления и программируется перед циклом формирования отверстий.
**Ряд отверстий** — HOLES1. С помощью этого цикла можно изготовить ряд отверстий, лежащих на одной прямой или решётку отверстий. Тип отверстий определяется через включенный до этого модально цикл сверления. Внутри цикла на основе фактической позиции плоскостных осей и геометрии ряда

отверстии определяется, начнётся ли обработка отверстий с последнего или с первого отверстия. После этого осуществляется подвод ускоренным ходом к позициям сверления. Формат цикла имеет сл. вид:
HOLES 1 (SPCA, SPCO, STA1, FDIS, DBH, NUM), где

- SPCA (Start Point of Cycle Abscissa) - абсцисса исходной точки на прямой (абс.)
- SPCO (Start Point of Cycle Ordinate) - ордината исходной точки на прямой (абс.). Исходная точка служит для определения расстояний между отверстиями. От этой точки указывается расстояние до первого отверстия FDIS
- STA1 - угол прямой к абсциссе, вводится в градусах. Диапазон значений: - 180 < STA 1 < = 180 градусов
- FDIS (Fiist Distance) расстояние от первого отверстия до исходной точки (вводится без знака)
- DBH (Distance Between Holes) - расстояние между двумя отверстиями (вводится без знака)
- NUM (NUM ber)-количество отверстий

Пример:



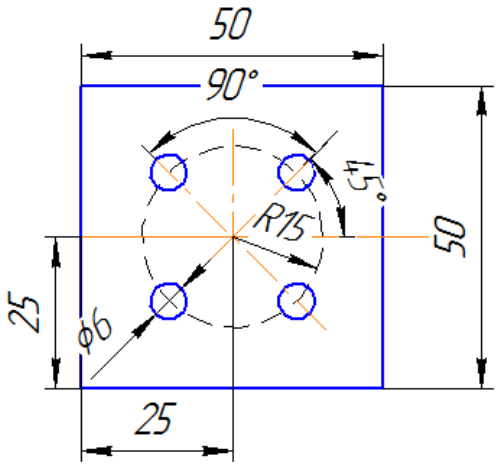

```
N1 G56 T1 D1 M6
N2 G0 X0 Y0 Z50 F150 M8
N3 S1000 M3
N4 MCALL CYCLE 81 (10, 0, 2, -12, 0)
N5 HOLES1 (20, -40, 15, 0, 30, 4)
N6 MCALL
N7 G0 Z50 M5 M9
N8 M30
```

**Окружность отверстий** - HOLES2. С помощью этого цикла можно изготовить ряд отверстий, лежащих на окружности. В цикле осуществляется последовательное прохождение позиций сверления отверстий с GO.
Формат цикла имеет сл. вид:
H0LES2 (CPA, CPO, RAD, STA 1 INDA, NUM), где

- CPA (Centre Point Abscissa) центр окружности отверстий по X (абс.)
- CPO (Centre Point Ordinate)- центр окружности отверстий по Y (абс.)

- RAD (RADius) -радиус окружности отверстий (вводится без знака)
- STA1 - начальный угол (угол поворота между полож отельным направлением абсциссы и первым отверстием). Диапазон значений: -180 < STA1 < = 180 градусов
- INDA (INDexing Angle) - угол повторного включения (угол поворота от одного отверстия к следующему). Если INDA=0, то угол повторного включения вычисляется внутри цикла из количества отверстий, располагая их равномерно на окружности.
- NUM (NUMber)- количество отверстий

Пример:



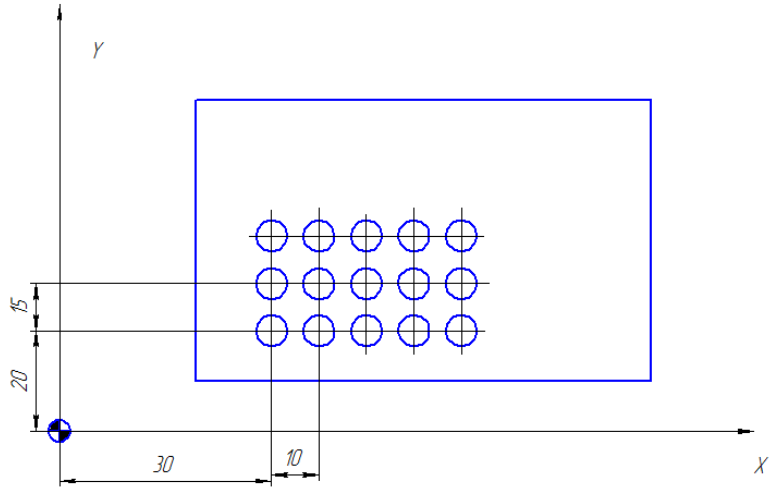

```
N1 G56 T1 D1 M6
N2 S1200 M3 F140
N3 G0 X25 Y25 Z10 M8
N4 MCALL CYCLE 81 (10, 0, 2, -15)
N5 HOLES2 (25, 25, 15, 45, 90, 4)
N6 MCALL
N7 G0 Z50 M5 M9
N8 M30
```

**Матрица отверстий** - CYCLE 801. С помощью этого цикла можно изготовить решётку отверстий. Цикл осуществляет последовательность обработки отверстий (или порядная или по столбцам) таким образом, чтобы как можно больше сократить холостые ходы между ними. Начальной позицией является соответственно одна из четырёх возможных угловых позиций. Формат цикла имеет сл. вид:

CYCLE 801 (SPCA, SPCO, STA1, D1S1, DIS2, NUM 1, NUM2), где

- SPCA - абсцисса первого отверстия (абс.)
- SPCO - ордината первого отверстия (абс.)
- Оба этих параметра определяют первую точку решётки отверстий. От этой точки указывается расстояние между рядами и столбцами
- STA 1 - угол решётки к абсциссе в градусах (любой угол в плоскости)
- DIS1, DIS2 - расстояние между столбцами и рядам и (вводятся без знака)
- NUM1, NUM2 - количество столбцов, рядов

Пример:




```
N1 G56 T1 D1 M6
N2 S1000 M3 F140
N3 G0 X0 Y0 Z50 M8
N4 MCALL CYCLE 81 (10, 0, 2, -12, 0)
N5 HOLES2 (30, 20, 0, 10.15, 5, 3)
N6 MCALL
N7 M9 G0 Z50 M5
N8 M30
```

## Фрезеровальные циклы

**Контурное фрезерование CYCLE 72.** С помощью цикла CYCLE72 осуществляется фрезерование контура, определённого в подпрограмме. Контур не обязательно должен быть замкнут и должен состоять как min из двух кадров (начальная и конечная точки), т.к. подпрограмма контура вызывается внутри цикла. В подпрограмме первым кадром должна быть запрограммирована стартовая точка контура. Цикл работает с коррекцией на радиус фрезы, которая включается в цикле, поэтому в подпрограмме не задаётся. Цикл применяется как для черновой, так и чистовой обработки. Исходной позицией является любая позиция, из которой можно без столкновений достичь начальной точки контура на высоте плоскости отвода. Формат цикла имеет сл. вид (параметры нельзя опускать):

CYCLE72 (KNAME, RTP, RFP, SDIS, DP, MID, FAL, FALD, FFP1, FFD, VARI, RL, AS1, LP1, FF3, AS2, LP2), где

- KNAME - имя подпрограммы контура заключается в кавычки “....” и она может находиться вне текущей программы, внутри текущей программы или в другой программе

Примеры вызова подпрограммы:

“KONTUR 1” - подпрограмма KONTUR_l находится вне текущей УП

“ABC: ENDE” - фрезеруемый контур находится в текущей УП и вызывается с кадра, помеченного меткой ABC и заканчивается кадром сметкой ENDE

KNAME = “/_N_SPF_DIR/_N_KONTUR_ 1_SPF:N30:N50” - фрезеруемый

контур определён в кадрах от N30 до N50 программы с именем KONTUR_1 (имя программы должно быть записано полностью с указанием пути и расширением). Если в УП KONTUR_1 были изменены номера кадров, то должны быть согласованы и номера кадров для сегмента в KNAME

- RTP, RFP, SDIS, DP - параметры как для CYCLE 81
- MID (Maximal Infeed Depth) - максимальная глубина фрезерования по оси Z (задаётся без знака). Врезание на глубину производится в цикле равными шагами. Цикл рассчитывает это врезание автоматически, исходя из параметров MID и DP, выбирается min возможное число шагов. При MID=0, врезание на всю глубину фрезерования.
- FAL - чистовой припуск по краю контура (X,Y), задаётся без знака
- FALD - чистовой припуск по дну контура (Z), задаётся без знака
- FFP1 (Feed For Plane) - рабочая подача для обработки поверхности
- FFD - подача для врезания на глубину (задаётся без знака)
- VARI - тип обработки (задаётся без знака). Возможные значения:

Разряд единиц:
1- черновая обработка
2- чистовая обработка
Разряд десятков:
0 - промежуточный путь с 00
1 - промежуточный путь с G1
Разряд сотен:
0 - отвод в конце ко тура до RTP
1- отвод в конце контура на RFP + SDIS
2- отвод в конце контура на SDIS
3-нет отвода а конце контура

- RL - обход контура но центру (G40), справа (G 42) или слева (G41), задаётся без знака
- AS1- траектория подвода инструмент к контуру (задаётся без знака).

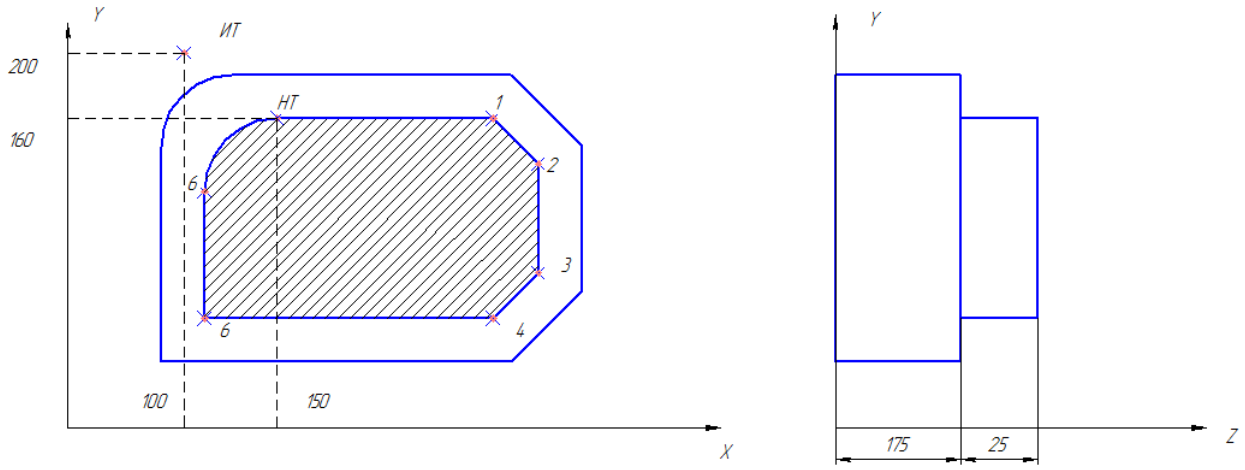
Возможные значения:
**Разрад единиц**: 1 -тангенциальная прямая (касательная) 2-четверть круга 3 - полукруг
**Разряд десятков**: 0- подвод к контуру а плоскости 1- подвод к контуру по пространственной траектории

- LP1 длина пути подвода (по прямой) или радиус подвода (для круга), т.е. расстояние от внешней кромки инструмента до стартовой точки контура (задаётся без знака), значение должно быть больше нуля.
- FF3 - подача отвода для промежуточных позиционирований в плоскости (свободный ход), если движения с G1. Если значение подачи не запрограммировано, то промежуточные движения при G1 осуществляются с подачей FFPI
- AS2 - траектория отвода инструмента от контура (задаётся без знака). Значения такие же, как для ASI. Бели AS2 не запрограммировано, то характеристика пути отвода идентична пути подвода ASI.

- LP2 - длина пути отвода (по прямой) или радиус отвода (для круга), задаётся без знака, значение должно быть больше нуля.

*Примечание:* При G40 путь подвода или отвода является расстоянием от центра инструмента до начальной или конечной точки контура и подвод/отвод возможен только по прямой

Пример: произвести черновое фрезерование замкнутого контура снаружи, промежуточное перемещение на G1, возврат по Z на RFP + SDIS




Параметры для вызова цикла:

1. Плоскость отвода (RTP) 250мм
2. Референтная плоскость (RFP) 200мм
3. Безопасное расстояние (SDIS) 3мм
4. Глубина фрезерования (DP) 175мм
5. max глубина врезания по Z (MID) 5мм
6. Чистовой припуск по кромке контура (FAL) 1мм
7. Чистовой припуск по глубине (FALD) 1,5мм
8. Подача плоскости (FFP1) 60мм/мни
9. Подача врезания на глубину (FFD) 40мм/мин
10. Тип обработки (VARI) см. условие 11
11. Обход контура слева (RL) G41
12. Подвод по дуге в четверть круга радиусом 20мм (AS1) 2
13. Радиус дуги подвода (LP1) 20мм
14. Подача обратного хода (FF3), промежуточный путь: 1000мм/мин
15. Отвод по дуге в четверть круга радиусом 20мм (AS2) 2
16. Радиус дуги отвода (LP2) 20мм

Управляющая программа (пример1):

N5 G54

N10 T2 D1 M6 - фреза концевая Ø25

N15 S300 M3 - программирование скорости вращения шпинделя

N20 G17 G90 G94 GO XI00 Y200 Z250 M8 - выход в исходную позицию

N25 CYC LE 72 (“MYKONTUR”, 250. 200, 3, 175,5. I, 1.5, 60, 40, 111, 41, 2, 20, 1000, 2, 20)- вызов цикла фрезерования

N30 X100 Y200 2250 M5 M9 - отвод, выключение шпинделя, СОЖ

N35 M30 - конец программы

MYKONTUR - подпрограмма фрезерования контура
N1 G90 G1 XI50 Y160 - перемещение в ИТ контура
N2 X230 CHF=10 - фрезерование поверхности (ИТ - 1 -2) со скосом 10мм
N3 Y80 CHF=10 - фрезерование поверхности (2 -3 - 4) со скосом 10мм
N4 X125 - фрезерование поверхности (4-5)
N5 Y135 - фрезерование поверхности (5 6)
N6 G2 XI50 Y160 CR=25 - фрезерование дуги радиусом 25мм (6 НТ)
N7 M17 - конец подпрограммы

**Пазы на окружности SLOT1.** Цикл SLOT1 это комбинированный цикл черновой - чистовой обработки и для него требуется фреза с режущим по центру торцовым зубом. С помощью этого цикла можно обрабатывать продольные пазы, расположенные на окружности радиально. Исходной позицией является любая позиция, из которой без столкновений можно достичь любого из продольных пазов. Формат цикла имеет сл. вид:
SLOT 1 (RTP, RFP, SDIS, DP, DPR, NUM, LENG, WID, CPA. СЮ, RAD, STAl, INDA, FFD, FFPl, MID, CDIR, FAL, VARI, MIDF, FFP2, SSF, FALD, STA2)

- RTP - SDIS - параметры аналогичны CYCLE 81
- DP, DPR - глубина продольного паза (вводится абс. или инкр. без знака)
- NUM - количество продольных пазов
- LENG - длина продольного паза (без знака). Если длина меньше диаметра фрезы, то цикл отменяется с ошибкой 61105 (слишком большой R- фрезы)
- WID (WIDth) - ширина паза (вводится без знака). Диаметр фрезы должен быть меньше ширины паза, иначе появляется ошибка 61105 (слишком большой R-фрезы, цикл отменяется) и не может быть меньше половины ширины паза (контроль не осуществляется)
- CPA, CPO, RAD - центр окружности X Y (абс.), радиус окружности (вводится без знака)
- STA 1 - начальный угол паза к положительной оси X в градусах
- INDA - угол повторного включения (угол поворота от одного паза к другому). Если INDA=0, то угол повторного включения вычисляется внутри цикла из количества пазов, располагая их равномерно на окружности.
- FFD, FFP1 - подачи на глубину врезания (Z), для обработки поверхности (X/Y)
- MID - max глубина врезания на проход по оси Z (без знака). Если M ID=0, то обработка за один проход
- CDIR (Cuttinq DIRection) - направление фрезерования для обработки паза. Значения:
- 0 - попутное фрезерование (по направлению вращения шпинделя)
- 1 - встречное фрезерование

- 2 - с G2 (независимо от направления вращения шпинделя)
- 3-с G3
- FAL (Finishing Allowance) - чистовой припуск по контуру паза (без знака). Если задано большее значение, чем оно может быть при данной ширине и используемой фрезе, то FAL автоматически уменьшается до максимально возможного значения. При черновой обработке в этом случае осуществляется маятниковое фрезерование с подачей на глубину на обеих конечных точках паза.
- VARI - режим обработки (вводится без знака).3начения:

**Разряд единиц:**

0 - комплексная обработка

1- черновая обработка

2- чистовая обработка

**Разряд десятков**:

1 - вертикальное врезание с G1

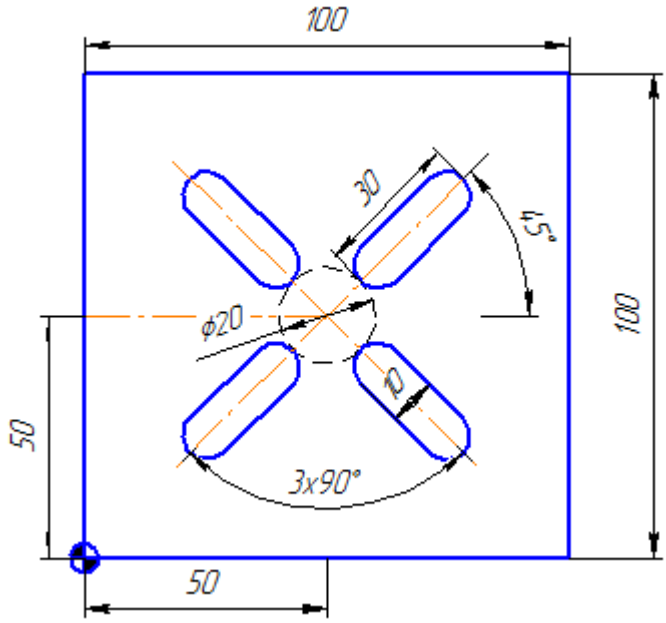
2- маятниковое врезание с G1

При задании VARI=30 на последней глубине черновой обработки осуществляется чистовая обработка края.

- MIDF (Maximum In feed Depth Finishing) - max глубина врезания для чистовой обработки, при MIDF 0 врезание до конечной глубины паза
- FFP2 (Feed For Plane) - подача для ЧИСТОВОЙ обработки, если FFP2=0, то действует подача FFPI
- SSF (Spindle Speed Finishing) - число оборотов дли чистовой обработки, если SSF-Ч), то действует число оборотов шпинделя, заданное перед циклом
- FALD - чистовой припуск на основании паза
- STA2 - угол врезания для маятникового движении. Подача программируется в FFD

*Примечание:* Перед вызовом цикла активировать коррекцию инструмент иначе следует отмена цикла с ошибкой 61000 (нет активной коррекции инструмента). Бели из-за неправильных значений параметров возникают взаимные повреждения контура пазов, то обработка цикла не начинается, выводится ошибка 61104 (повреждение контура пазов).

Пример:

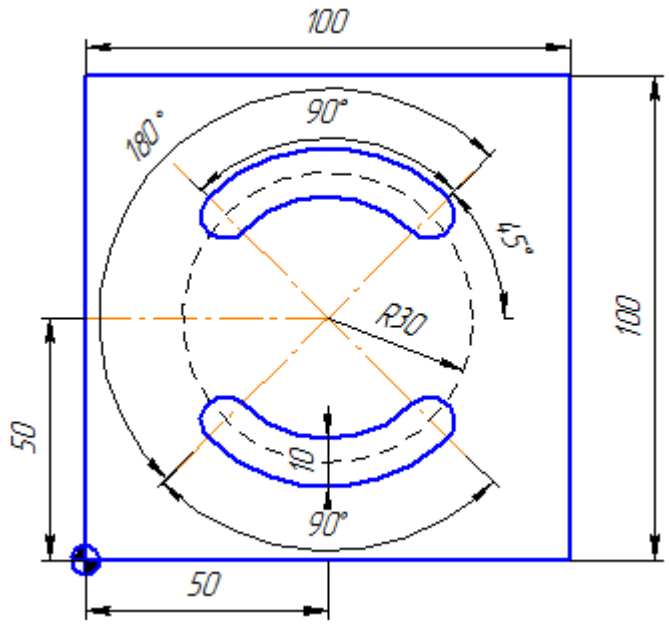



N1 G54 T1 D1 M6
N2 S1000 M3
N3 G0 X50 Y50 Z2
N4 SLOT 1 (2, 0, 1, -6, 0, 4, 30, 10, 50, 50, 10, 45, 90, 80, 350, 0, 3, 0.2, 0, 6, 400, 1500, 0.5, 5)
N5 G0 Z50 M5
N6 M30

**Кольцевая канавка SLOT2.** Цикл SLOT2 это комбинированный цикл черновой - чистовой обработки и для него требуется фреза с режущим по центру торцовым зубом. С помощью этого цикла можно обрабатывать кольцевые канавки, расположенные на окружности. Исходной позицией является любая позиция, из которой без столкновений можно достичь любой из канавок. Переход от одной канавки к другой осуществляется либо по прямой с G0, либо по круговой траектории с запрограммированной в FFCP подачей (FFCP устанавливается в цикле, начиная с версии ПО 6.3)
Формат цикла имеет сл. вид:
SLOT 2 (RTP, RFP, SDIS, DP, DPR, NUM, AFSL, WID, CPA, CTO, RAD, STA1, INDA, FFD, FFP1, MID, CDIR, FAL, VARI, MIDF, FFP2, SSF), где

- AFSL (Angle For Slot Length) - угол кольце вой канавки (без знака)
- CDIR - направл. фрезеров, для обработки канавки. Значения: 2 - для G2, 3 для G3
- VARI - режим обработки (без знака). Значения: 0 - комплексная обработка 1 черновая обработка 2 - чистовая обработка
- Остальные параметры аналогичны циклу SLOT1

Пример:




N1 G54 T1 D1 M6
N2 S1500 M3
N3 G0 X50 Y50 Z2
N4 SLOT 2 (2, 0, 1, -6, 0, 2, 90, 10, 50, 50, 30, 45, 180, 80, 100, 2, 3, 0.2, 0, 6, 200, 2500)
N5 G0 Z50 M5
N6 M30

**Фрезерование прямоугольного кармана POCKET3** (цикл доступен от версии ПО 4). Цикл может использоваться для черновой и чистовой обработки. Для чистовой обработки необходима торцовая фреза, врезание на глубину всегда осуществляется в центре кармана, поэтому рекомендуется предварительно там произвести сверление. Исходной позицией является любая позиция, из которой без столкновений можно достичь центра кармана на высоте плоскости отвода.

Формат цикла имеет сл. вид:

POCKET 3 (RTP, RFP, SDIS, DP, LENG, WID, CRAD, PA, PO, STA, MID, FAL, FALD, FFP1, FFD, CDIR, VARI, MIDA, AP1, AP2, AD, RAD1, DP1), где

- Параметры RTP- DP аналогичны циклу CYCLE 81.
- LENG, WID, CRAD - длина, ширина кармана и угловой радиус. Карман может быть измерен от центра или от угловой точки. При измерении от угла LENG и WID вводятся со знаками, благодаря чему положение кармана определяется однозначно
- PA, PO - исходная точка кармана (абс.) по оси X Y (либо центр, либо одна из угловых точек)
- MID - max глубина врезания на проход по оси Z (без знака). Если M ID=0, то обработка за один проход
- FAL- чистовой припуск на краю кармана (X, Y), вводится без знака

- FALD - чистовой припуск на основании, вводится без знака (POCKET1 не имеет чистового припуска на основании)
- FFD, FFP1 - подача на глубину и в плоскости
- CDIR - направление фрезерования (без знака). Значения:

0- попутное фрезерование, если направление вращения шпинделя по часовой стрелке (M3)

1- встречное фрезерование, если направление вращения шпинделя против часовой стрелки (M4)

2- с G2 (по часовой стрелке) независимо от направления вращения шпинделя

3- с G3 (против часовой стрелки) независимо от направления вращения шпинделя

- VARI - режим обработки (без знака). Значения:

**Разряд единиц:**

1- черновая обработка

2- чистовая обработка

**Разряд десятков:**

0 -вертикальное врезание по центру кармана с G0

1- вертикальное врезание по центру кармана с G1

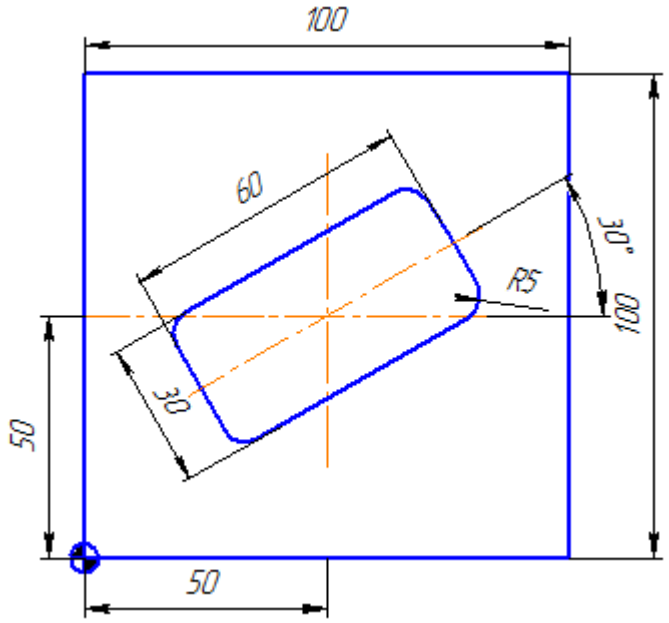
2 -врезание по спиральной траектории, определяемой через радиус RADI и глубину на оборот DPI (глубина врезания рассчитывается как max глубина и всегда включает целое число оборотов спиральной траектории)

3- маятниковое врезание с G1. Максимальный угол врезания программируется в RAD1, длина пути вычисляется внутри цикла, подача программируется в параметре FFD

Следующие параметры могут задаваться по выбору. Они определяют стратегию врезания и перекрытие при выборке (вводятся без знака).

- MIDA - max ширина фрезерования при выборке в плоскости. Если параметр МЮА=0 (или не запрограммирован), то цикл использует 80% диаметра фрезы как max ширину фрезерования.
- AP1, AP2, AD - черновые размеры кармана по длине, ширине и глубине, например, предварительно отлитых деталей или штамповки
- RAD1 - радиус спиральной траектории при врезании (относительно траектории центра инструмента) или максимальный угол врезания для маятникового движения
- DP1 - глубина врезания на оборот (360°) по спиральной траектории

Пример:




```
N1 G54 T1 D1 M6
N2 S1000 M3
N3 G0 X50 Y50 Z2
N4 POCKET 3 (2, 0, 1, -6, 60, 30, 5, 50, 50, 30, 2, 0.2, 0.1, 150, 80, 3, 21, 0, 0, 0, 0, 10, 2)
N5 G0 Z50 M5
N6 M30
```

**Фрезерование кругового кармана POCKET 4** (цикл доступен от версии ПО 4). Цикл может использоваться для черновой и чистовой обработки. Для чистовой обработки необходима торцовая фреза, врезание на глубину всегда начинается с центра кармана, поэтому рекомендуется предварительно там произвести сверление. Исходной позицией является любая позиция, из которой без столкновений можно достичь центра кармана на высоте плоскости отвода. Формат цикла имеет сл. вид:

POCKET 4 (RTP, RFP, SDIS, DP, PRAD, PA, PO, MID, FAL, FALD, FFP1, FFD, CDIR, VARI, MIDA, AP1, AD, RAD1, DP1), где

- Параметры RTP- DP аналогичны циклу CYCLE 81
- PRAD - радиус кармана
- PA, PO - центр кармана (абс.)
- VARI - режим обработки (без знака). Возможные значения:

**Разряд единиц:**

1- черновая обработка

2- чистовая обработка

**Разряд десятков:**

0 - вертикальное врезание по центру кармана с G0

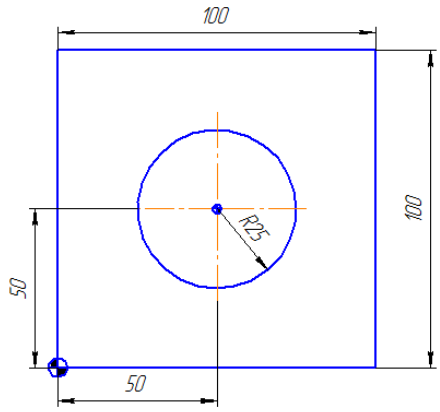
1 - вертикальное врезание по центру кармана с G1

2 - врезание по спиральной траектории

- AP1 – черновой размер радиуса кармана
- RAD1 – радиус спиральной траектории при врезании (относительно траектории центра инструмента)
- MID, FAL, FALD, FFP1, FFD, CDIR, MIDA, AD, DP1 – см. POCKET 3

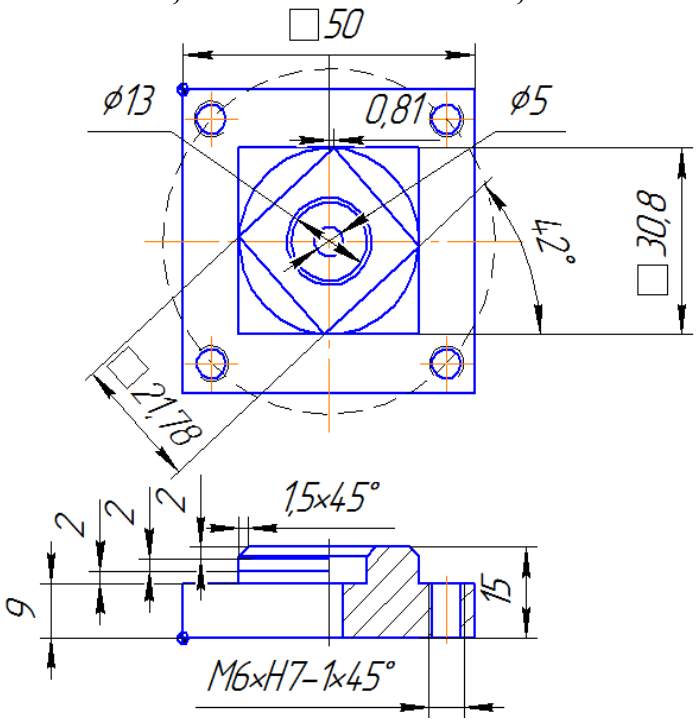
Параметры MIDA, AP1, AD, RAD1, DP1 могут программироваться по выбору. При черновой обработке только края кармана (глубина не обрабатывается), параметр AD должен быть равен или больше полученного значения AD ≥ DP – FALD

Пример:




```
N1 G54 T1 D1 M6
N2 S1000 M3
N3 G0 X0 Y0 Z2
N4 POCKET 3 (2, 0, 1, -6, 25, 0, 0, 3, 0.2, 0.1, 150, 80, 3, 21, 7, 0, 0, 10, 3)
N5 G0 Z50 M5
N6 M30
```

**Корпус** 260-**1015.130, заготовка 50x50x15, сплав** Al Cu Pb Bi



```
MAINPROGRAM_MPF
N1 G54 G64
N2 TRANS X25 Y-25 Z15
;FACE MILL 40mm
N3 T1 D1 M6
N4 S1300 F350 M3
N5 G0 X-47 Y0 Z2
N6 G1 Z0
N7 G1 X47
N8 G0 Z10
;SHANK END MILL 10mm
N9 T2 D1 M6
N10 S2800 F350 M3
N11 G0 X-32 Y0 Z2
N12 Z-5.9
N13 SQUARE
N14 G0 Z10
;SLOT END MILL
N15 T3 D1 M6
N16 S4000 F600 M3
N17 G0 Z2
N18 Z-6
N19 SQUARE
N20 G0 Z2
N21 Z-4
N22 CIRCLE
N23 G0 Z2
N24 AROT Z42
N25 X-32 Y0
N26 Y-2
N27 SQUARE2
N28 AROT Z-42
N29 G0 Z10
N30 X0 Y0 Z2
N31 POCKET
N32 X0 Y0 Z10
;NC DRILL
N33 T5 D1 M6
N34 S2500 F80 M3
N35 G0 Z2
N36 MCALL CYCLE81(2,-6,7,,1.92)
N37 HOLES2(,,28.5,135,90,4)
```

```
N38 MCALL
N39 G0 X0 Y0 Z2
N40 CYCLE81(2,-6,7,,1.92)
N41 Z10
;TWIST DRILL 5mm
N42 T6 D1 M6
N43 S2500 F100 M3
N44 G0 X0 Y0 Z2
N45 MCALL CYCLE81(2,-6,1,,12)
N46 HOLES2(,,28.5,135,90,4)
N47 MCALL
N48 G0 X0 Y0 Z2
N49 CYCLE81(2,-6,1,,12)
N50 Z10
;ANGLE MILL 16X45^0
N51 T4 D1 M6
N52 S4500 F600 M3
N53 X-32 Y0
N54 Z-0.5
N55 AROT Z42
N56 SQUARE2
N57 AROT Z-42
N58 Z10
N59 G0 X0 Y0
N60 Z-0.5
N61 POCKET2(2,,1,,0.5,6.5,,,80,600,0.5,3,0,2,2,600,3500)
N62 Z10
;TAPPING M6
N63 T7 D1 M6
N64 S600 M3
N65 MCALL CYCLE840(2,-6,8,,12,0,4,3,0,0,1)
N66 HOLES2(,,28.5,135,90,4)
N67 MCALL
N68 Z50
N69 M30

SQUARE_SPF
N1 G1 G41 X-19.4 Y0 G247 DISR=5
N2 Y19.4
N3 X19.4
N4 Y-19.4
N5 X-19.4
N6 Y0 X-19.4
N7 G1 G40 X-32 Y0 G248 DISR=5
```

```
N8 G1 G41 X-15.4 Y0 G247 DISR=5
N9 Y15.4
N10 X15.4
N11 Y-15.4
N12 X-15.4
N13 Y0
N14 G40 X-32 Y0 G248 DISR=5
N15 M17

CIRCLE_SPF
N1 G1 G41 X-15.4 Y0 G247 DISR=5
N2 G2 X15.4 Y0 CR=15.4
N3 X-15.4 Y0 CR=15.4
N4 G1 G40 X-32 Y0 G248 DISR=5
N5 M17

SQUARE2_SPF
N1 G64
N2 G1 G41 X-10.89 Y0 G247 DISR=5
N3 Y10.89
N4 X10.89
N5 Y-10.89
N6 X-10.89
N7 Y0
N8 G1 G40 X-25 Y0 G248 DISR=5
N9 M17

POCKET_SPF
N1 POCKET2(2,,1,,6,6.5,,,50,200,3,3,0,2,3,200,2800)
N2 M17
```

## 5. **ЛАБОРАТОРНАЯ РАБОТА** SCADA-**СИСТЕМА**
## **«Создание проекта в Citect. Установление связей с ПЛК»**

**Цель работы.**

1. Научиться создавать проект в SCADA Citect на примере дорожного светофора.
2. Получить практические навыки в организации обмена между ПЛК и Scada системой.

**Задание.**

1. Создать управление светофором в Citect. Все сервера и элементы, входящие в проект должны иметь в своем названии ваш номер по журналу группы.
2. Организовать обмен информацией между Scada системой и ПЛК, установленном на стенде.
3. Создать необходимые переменные тэги для работы тестовой страницы «Светофор».
4. По результатам выполнения лабораторной работы подготовьте отчет содержащий настройки проекта, список переменных тэгов и ответы на контрольные вопросы.

Перед выполнением лабораторной работы загрузите в ПЛК MITSUBISHI программу, написанную на языке РКС.

**Последовательность выполнения лабораторной работы.**

Ниже приводится пошаговое описание действий по созданию проекта и установлению связей SCADA – ПЛК. Проект состоит из 3 этапов.

**ЭТАП 1** Создание проекта.

1. Запустить проводник Citect.

2. В проводнике Citect выбрать меню «Файл – Новый проект». Затем в появившемся окне заполнить название проекта и описание. Название должно быть уникальным, написано латиницей. Использоваться могут любые символы, за исключением точки с запятой (;), или одинарной кавычки (‘).

В поле «Описание» ввести назначение проекта и свой номер по журналу (смотри рисунок 5.1).

В поле «Путь» указать каталог, в котором будет храниться проект.

В поле «Стиль шаблона» выбрать стиль графических страниц, используемый по умолчанию. При желании его можно будет изменить на вкладке «Свойства страницы».

«Разрешение шаблона»: Разрешающая способность экрана для отображения стандартных графических страниц .

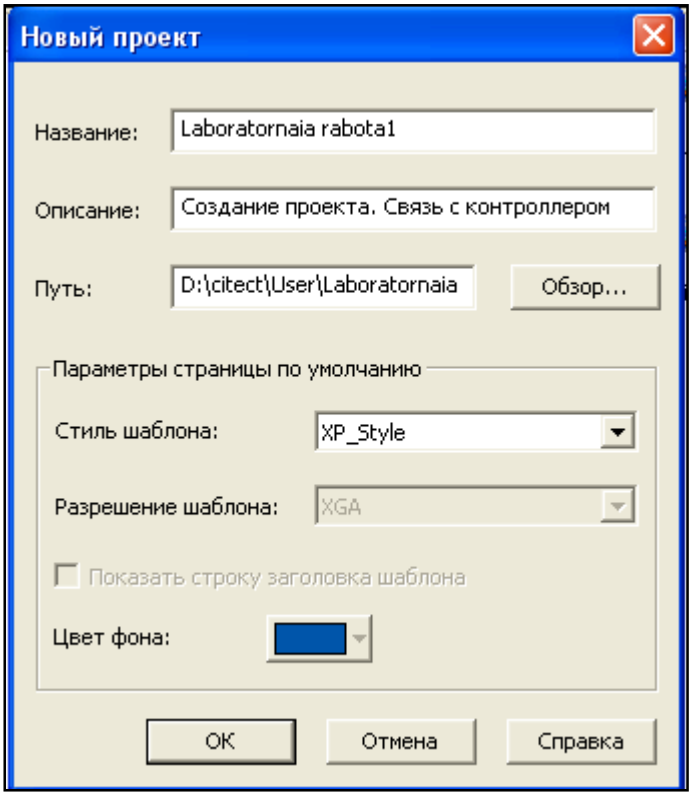



Рисунок 5.1 – Создание проекта

3. Создать кластеры, сервер ввода/вывода, сервер трендов и алармов, сервер отчетов.

Начинать с кластера. Кластер определяет, как и где будут работать различные сервера системы (ввода-вывода, алармов, трендов) и как они будут

взаимодействовать друг с другом. В каждой системе Citect должен быть хоть один кластер, и сервер ввода-вывода должен быть с ним связан. Для того чтобы создать кластер, открыть «Редактор проектов». Выбрать меню «Сервера – Кластеры». В появившемся окне ввести имя кластера (например MyCluster), его описание, и  нажать «Добавить» (смотри рисунок 5.2).

Следующим шагом прописать сетевой адрес системы, в данном случае компьютер работает не в сети. Поэтому прописать IP адрес 127.0.0.1. Если бы создавался проект как сетевое решение, то нужно было бы установить его IP адрес.

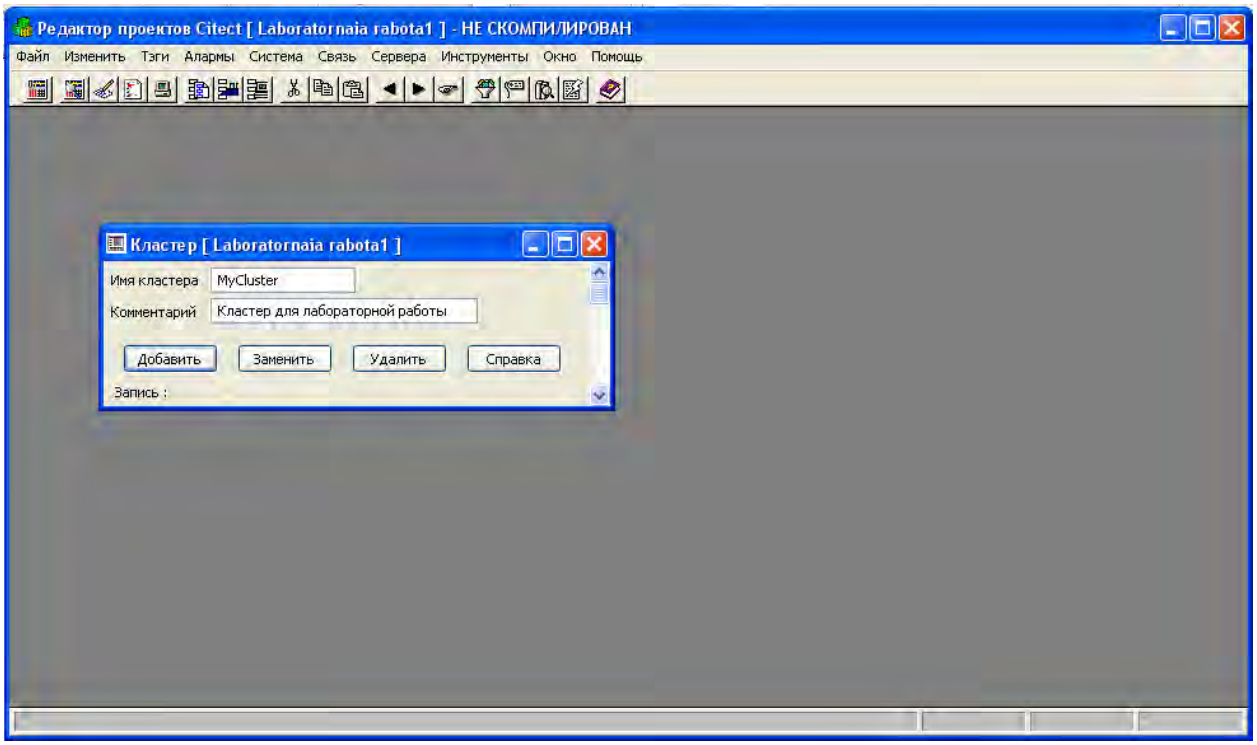



Рисунок 5.2 – Создание кластера

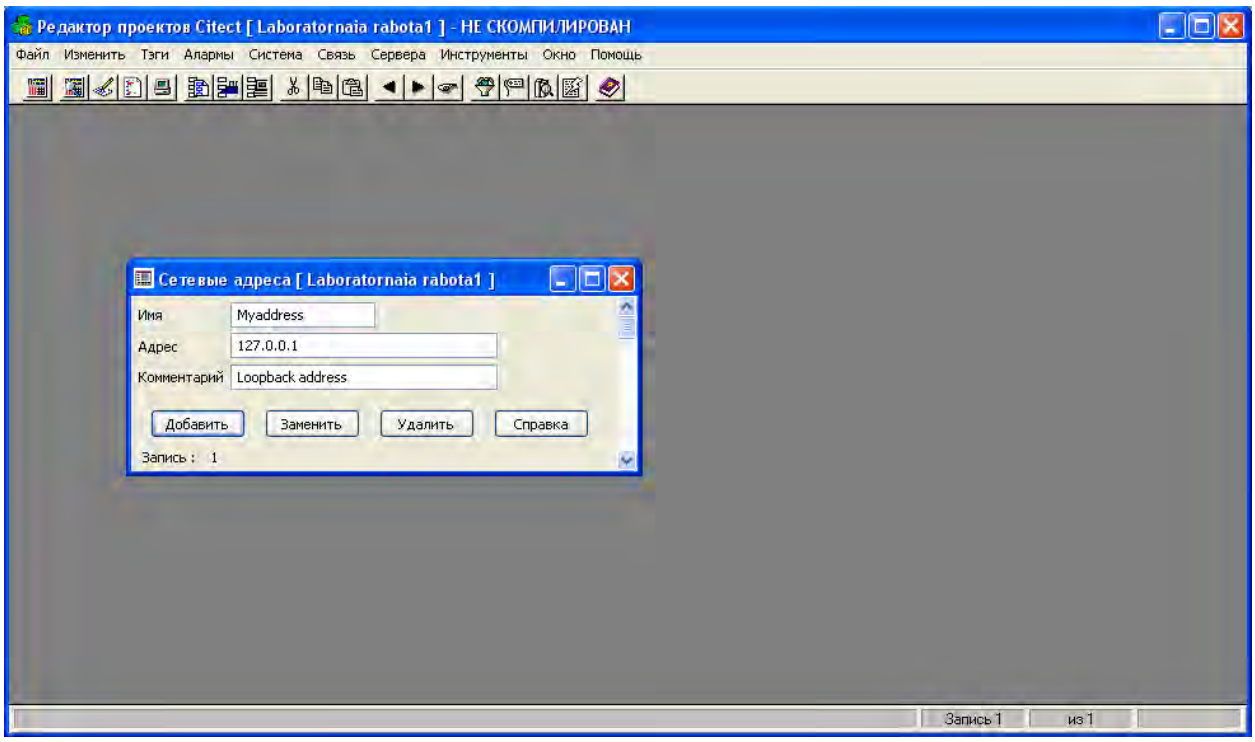



Рисунок 5.3 – Создание сетевого адреса

Для записи сетевого адреса выбрать вкладку «Сервера – Сетевые адреса». В появившемся окне внести необходимые параметры: имя сервера – Myaddress, адрес – 127.0.0.1, и комментарий (смотри рисунок 5.3).

После того как создан кластер и сетевой адрес, можно приступать к созданию серверов.

Для этого выбрать вкладку «Сервера – Сервер ввода/вывода». В появившемся окне прописать имя сервера, выбрать из списка кластер и сетевой адрес. Поля «Порт» и «Равноправный порт» оставить не заполненными. После этого нажать кнопку «Добавить».

Затем аналогичным образом создать сервера алармов, трендов и отчетов. Для всех соответствующих серверов нужно выбрать режим «Primary». Заполнение форм представлено на рисунках 5.4-5.7

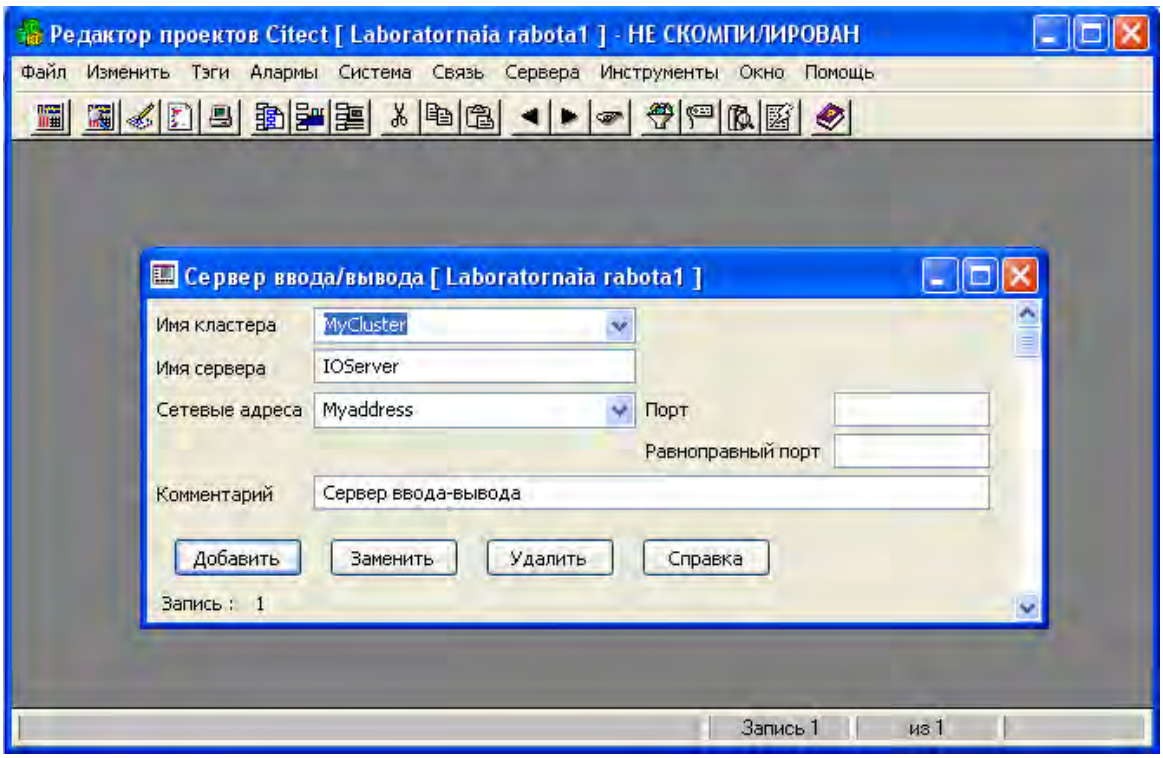



Рисунок 5.4 – Создание сервера ввода-вывода

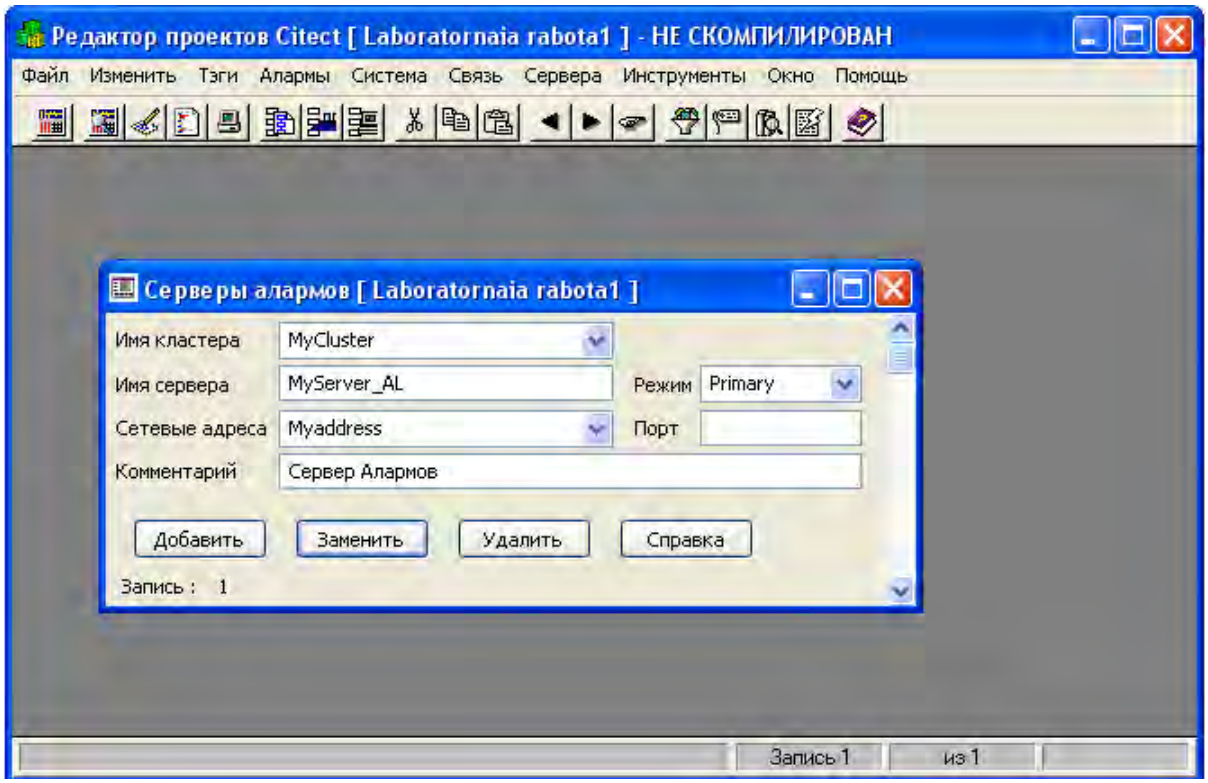
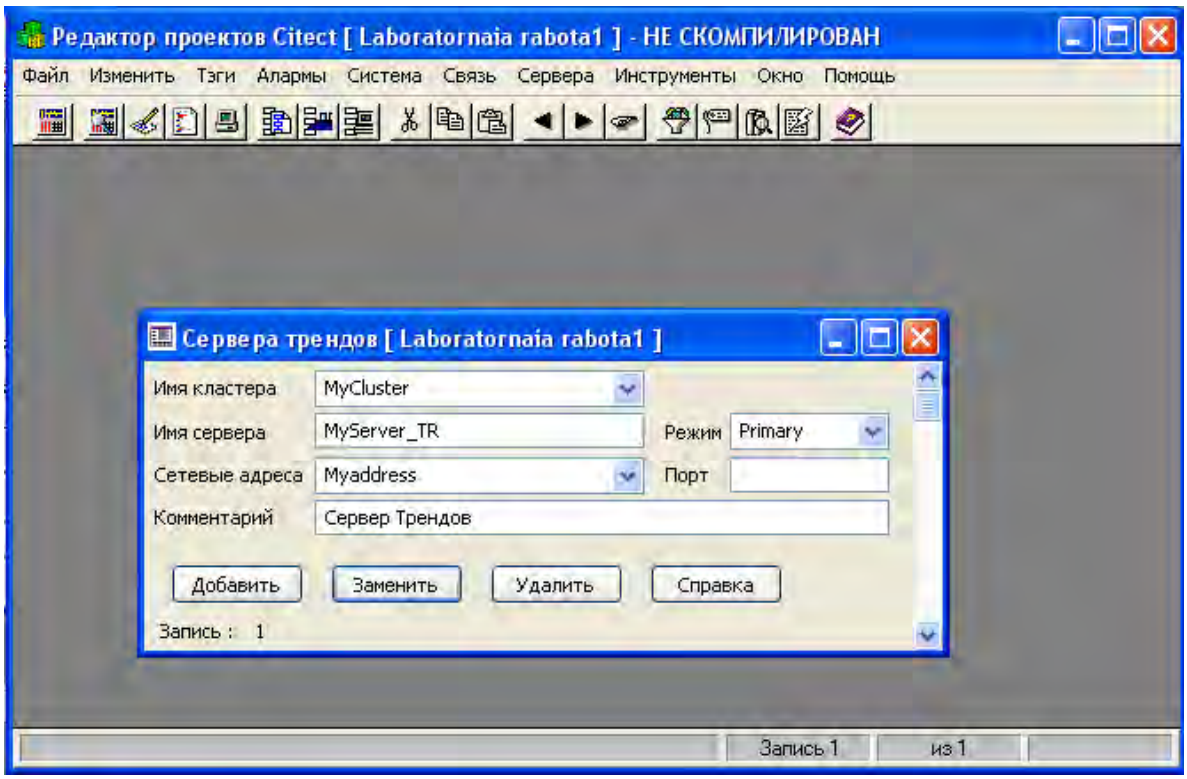



Рисунок 5.5 – Создание сервера алармов




\

Рисунок 5.6 – Создание сервера трендов

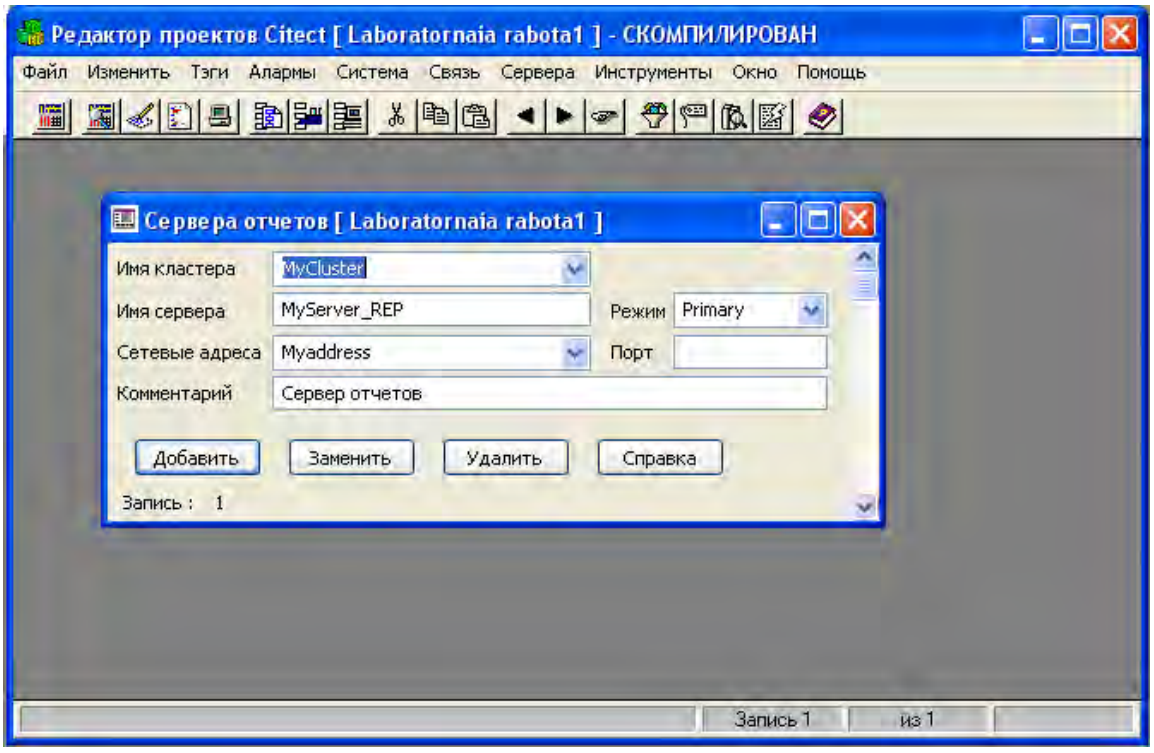



Рисунок 5.7 – Создание сервера отчетов.

**ЭТАП 2** Создание связи с контроллером

Для создания связи с контроллером выбрать вкладку в «Проводнике Citect» «Устройство связи – Экспресс настройка устройства ввода-вывода». Далее будет запущен «Мастер экспресс установки связи», который может сконфигурировать систему. В первом окне выбрать «Далее» (смотри рисунок 5.8).

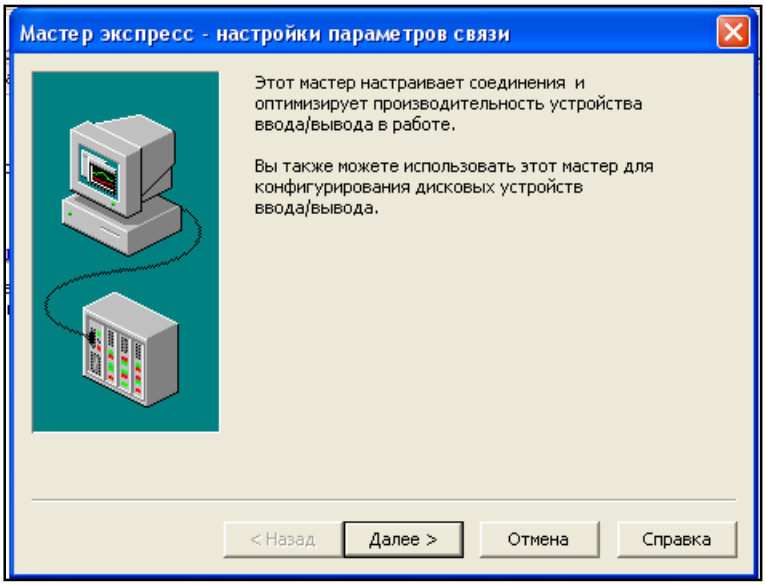



Рисунок 5.8 – Запуск Мастера настройки параметров связи

В следующем окне можно выбрать либо создать новый сервер ввода-вывода. Выбрать «Использовать существующий сервер ввода-вывода», и нажать «Далее» (смотри рисунок 5.9).

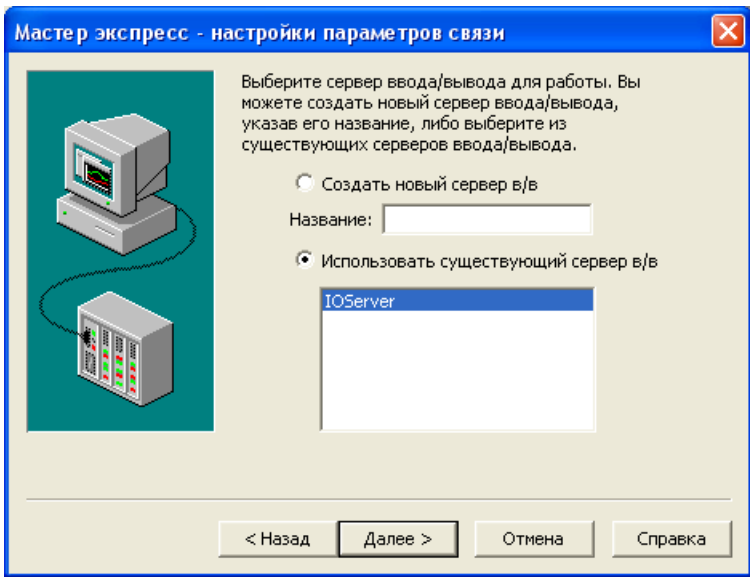



Рисунок 5.9 – Выбор сервера ввода-вывода

На следующем экране можно выбрать либо создать новое устройство ввода-вывода. Создать устройство IODev1, и нажать «Далее»(смотри рисунок 5.10).

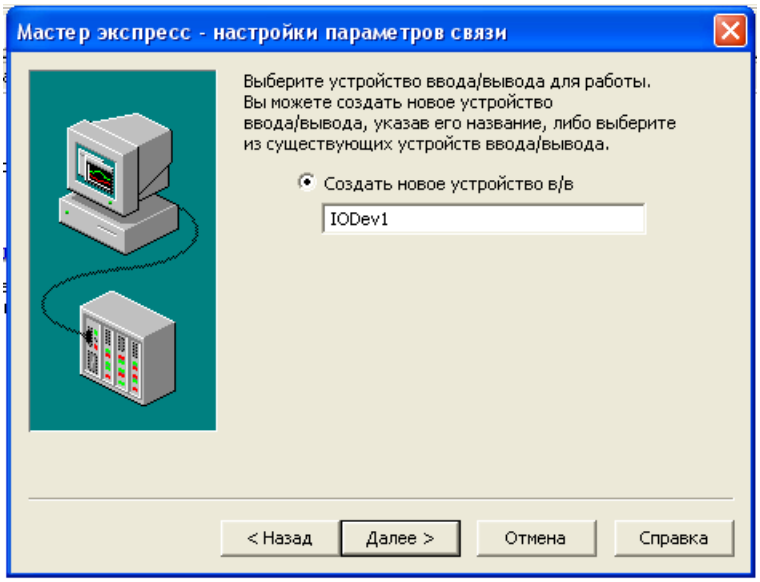



Рисунок 5.10 – Создание нового устройства ввода-вывода

Далее выбрать тип устройства ввода-вывода. Это может быть либо «Внешнее устройство ввода-вывода» (ПЛК, ПЧ, расходомер и т.д.), либо виртуальное устройство в памяти компьютера, которое используется для имитации реальных устройств (смотри рисунок 511).

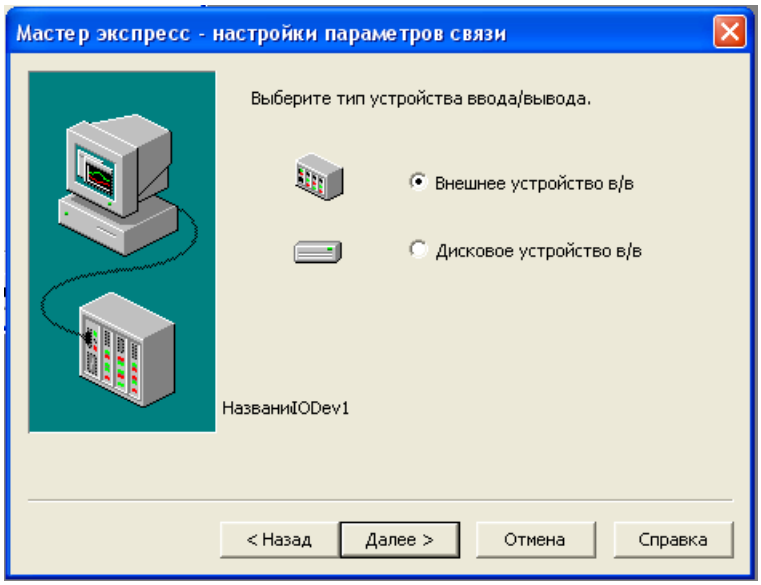



Рисунок 5.11 – Тип устройства ввода-вывода

Затем выбрать производителя устройства модель и способ подключения. Производитель – Mitsubishi, модель – Melsec-FX3U, тип подключения – Serial(Point to Point) (COM порт) (смотри рисунок 5.12).

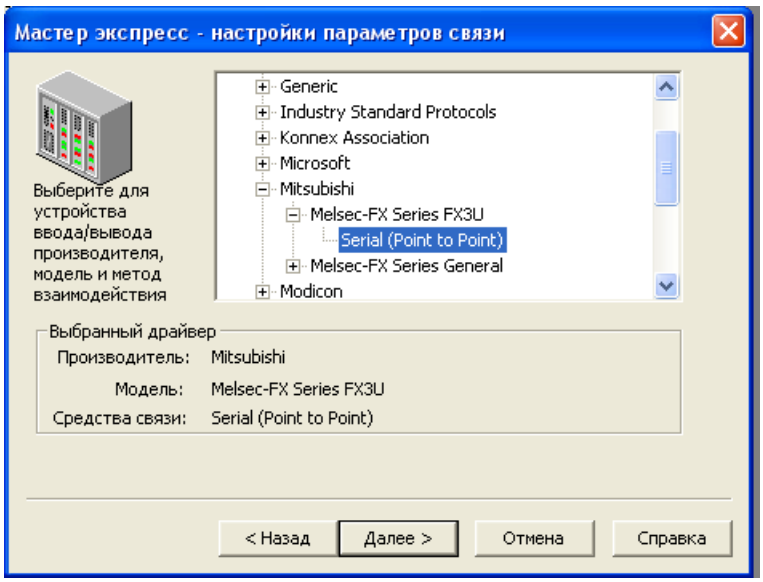



Рисунок 5.12 – Выбор драйвера для подключения к ПЛК.

При использовании GX-Simulator в качестве адреса устанавливаем Logical Station Number.

Далее система попросит указать адрес устройства, в данном случае ничего не менять, нажать «Далее» (смотри рисунок 5.13).

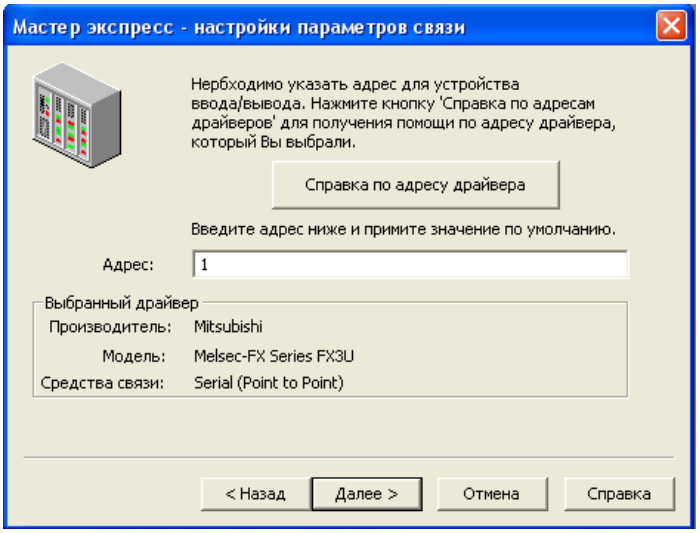



Рисунок 5.13 – Адрес устройства

Далее следует настройка подключения устройства через модем. Т.к. в проекте модем не используется, нажмите «Далее» (смотри рисунок 5.14).

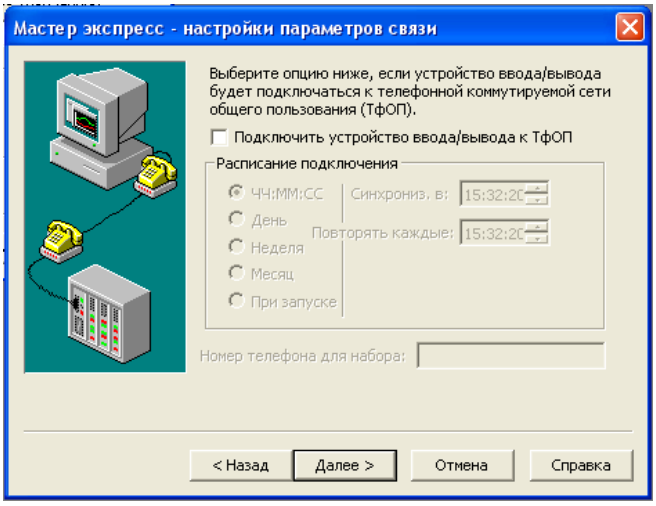



Рисунок 5.14 – Настройка подключения модема

Далее система предложит выбрать COM порт, к которому подключается ПЛК. В данном случае это – COM 8. В реальности номер COM порта может быть другим (смотри рисунок 5.15). Контроллер может подключаться с помощью кабеля программирования SC-09 (с преобразователем RS232-RS422), через порт программирования к COM порту компьютера. Либо с помощью простого COM кабеля через модуль расширения RS232 BD, к COM порту компьютера.

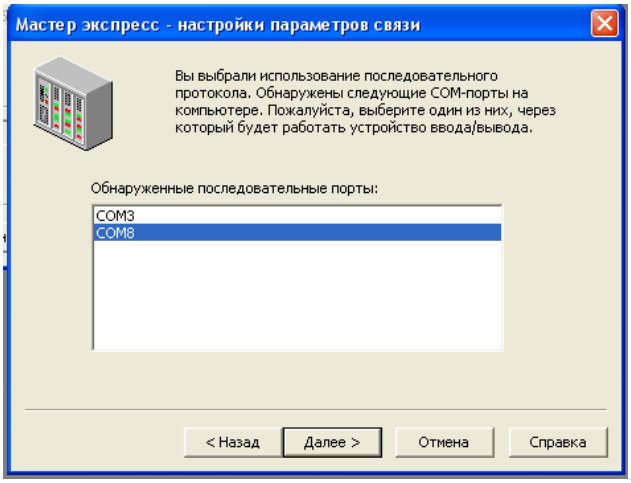



Рисунок 5.15 – Выбор COM порта для подключения ПЛК

Затем выбрать способ подключения к внешней базе данных тэгов, поэтому нажмите «Далее» (смотри рисунки 5.16 – 5.17)

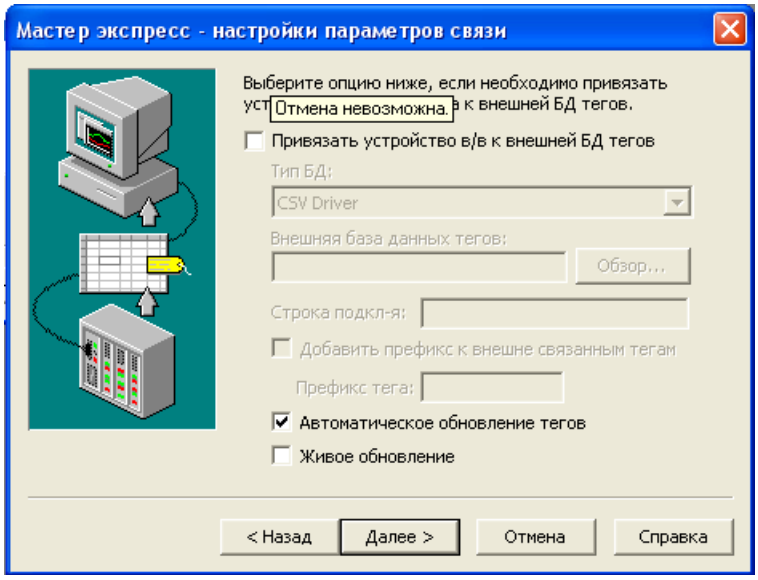



Рисунок 5.16 – Связь тэгов с внешней базой данных

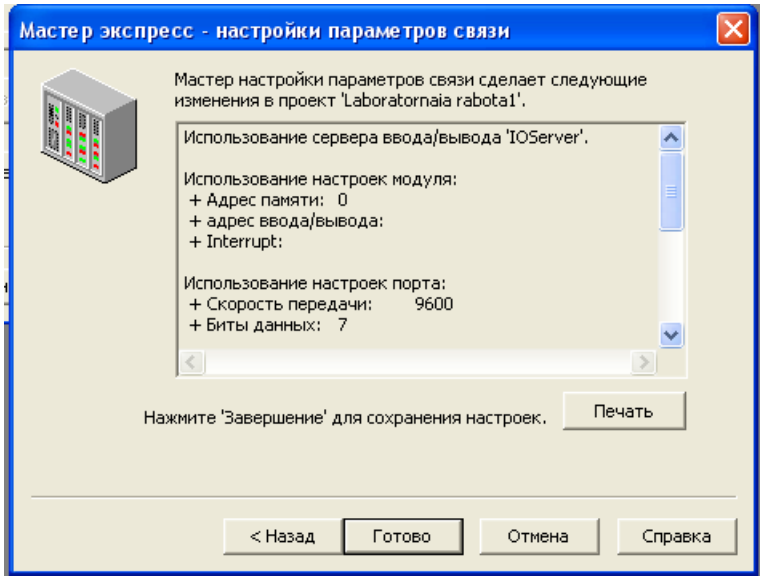



Рисунок 5.17 – Подтверждение выполнения настройки устройства связи

На завершающем этапе проверить выполненные настройки, и нажать «Готово».

На рисунке 5.18 показаны результаты работы мастера настройки устройств ввода-вывода. Он создал модуль, порт и устройство ввода-вывода .

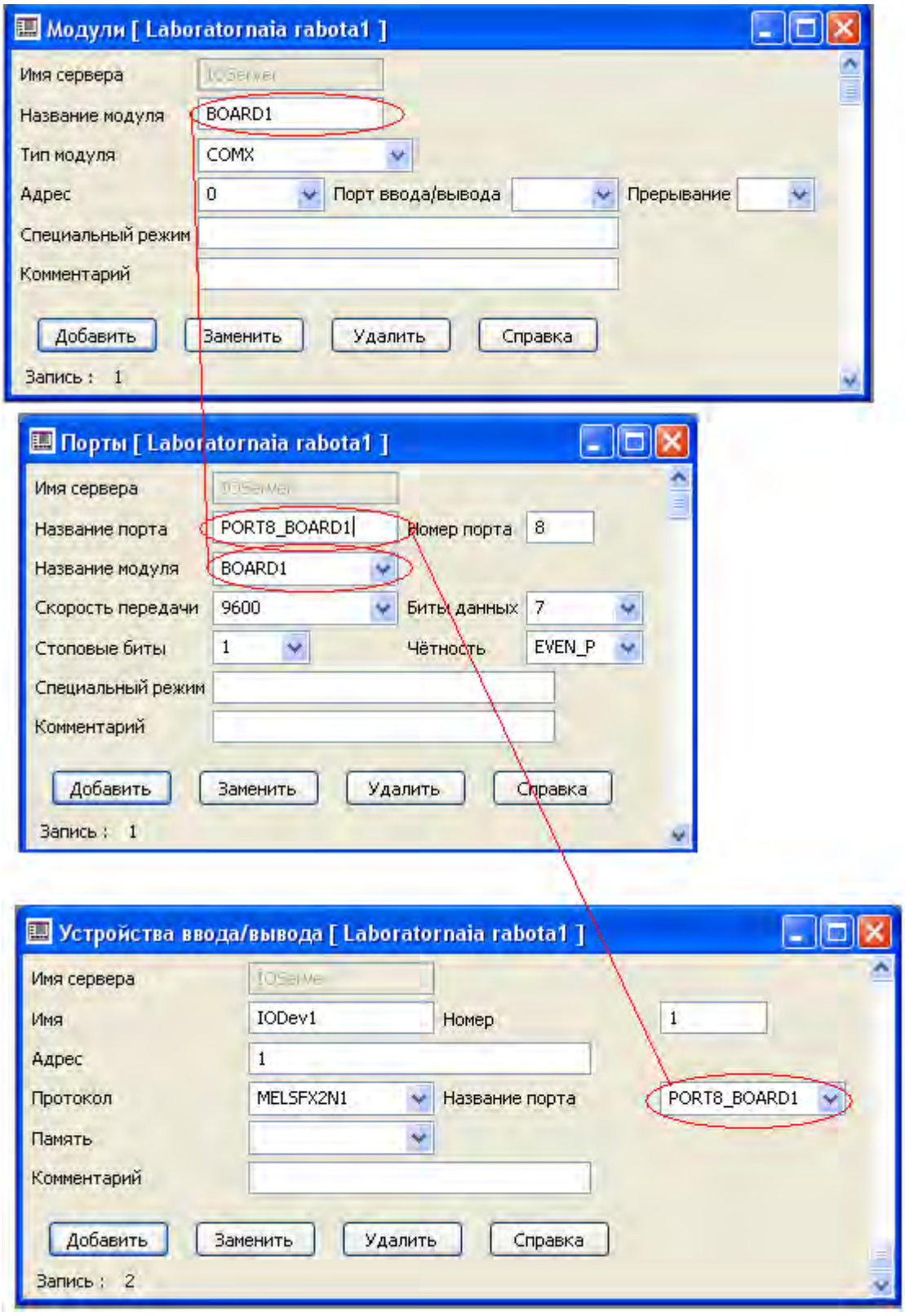



Рисунок 5.18 – Схема настройки параметров устройства ввода-вывода.

**ЭТАП 3**. Создание переменных тэгов.

Для того, чтобы проверить работоспособность установленной связи, создайте несколько тэгов и проверьте работу анимации на тестовой странице. Тэги и их параметры приведены в таблице 5.1.

Таблица 5.1 Тэги для лабораторной работы №1

| Название | Адрес | Тип данных | Устройство ввода-вывода | Дополнительные параметры |
|---|---|---|---|---|
| RED_COLOUR | M2 | Digital | IODev1 | - |
| YELLOW_COLOUR | M3 | Digital | IODev1 | - |
| GREEN_COLOUR | M4 | Digital | IODev1 | - |
| Timer | D0 | Int | IODev1 | Формат ## |
| Time_stop | D1 | Int | IODev1 | Формат ## |

Для того, чтобы создать тэги в «Редакторе проектов», выберите меню «Тэги — Переменные тэги», заполните поля параметров тэга в появившейся форме, затем нажмите кнопку «Добавить». Этим добавляется запись в базу данных тэгов проекта. Будьте внимательны при вводе параметров. Опечатка в имени тэга приведет к сбою в работе проекта. Если тэг уже создан, а вы редактируете его свойства, то по окончанию редактирования нажмите кнопку «Заменить». На рисунке 5.19 приведен пример заполнения формы свойств тэга.

После ввода всех тэгов создайте графическую страницу с произвольно выбранным оформлением – «Графика – Страницы – Создать новую страницу», сохраните страницу как Page1. Затем создайте изображение дорожного светофора (рисунок 5.20) и для отображения цветов в правильном порядке связать круги светофора, отображающие тот или иной свет, с соответствующими тэгами.

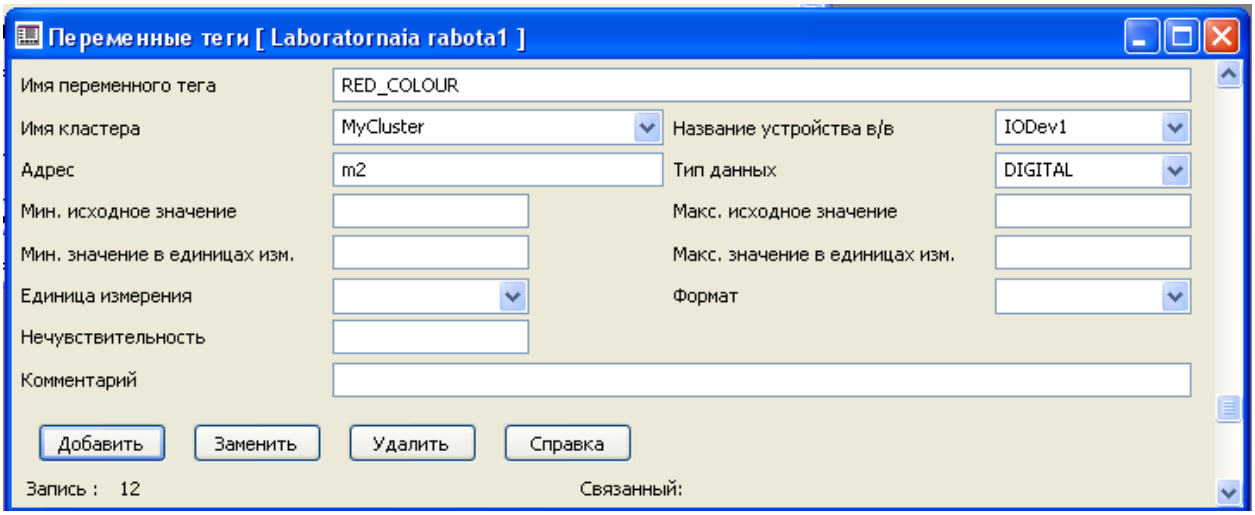



Рисунок 5.19 – Пример заполнения формы свойств тэга.

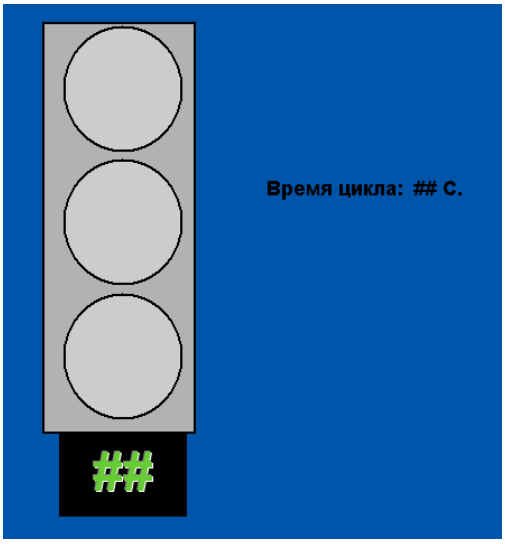



Рисунок 5.20 – Экран тестового проекта.

Для рисования светофора, используют не сложные инструменты, подобные обычно применяют в стандартных графических пакетах. Нарисуйте основной прямоугольник. Для этого нужно выбрать инструмент «Прямоугольник», кликнуть в произвольном месте страницы и, не отпуская клавиши, растянуть объект до необходимых размеров. После таких действий появится меню «Свойства: Прямоугольник», где можно во вкладке «Вид»

изменить ширину контурной линии, в поле «Заполнение» поставить галочку заполненный и выбрать цвет заполнения (смотри рисунок 5.21).

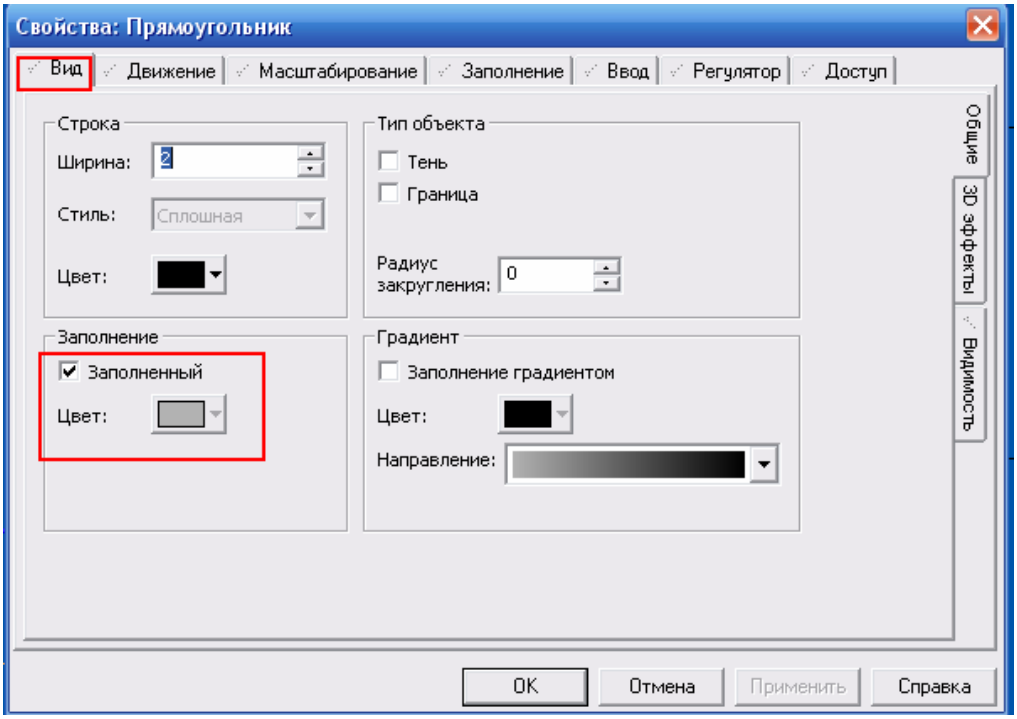



Рисунок 5.21 – Свойства объекта «Прямоугольник».

Выберите инструмент «Эллипс» и создайте окружность такого радиуса, чтобы она легко вписалась в ранее созданный прямоугольник. Построение аналогично тому, как строили прямоугольник. В свойствах объекта «Эллипс» необходимо во вкладке «Заполнение» выбрать управляющий тэг соответствующий красному цвету (смотри рисунок 5.22).

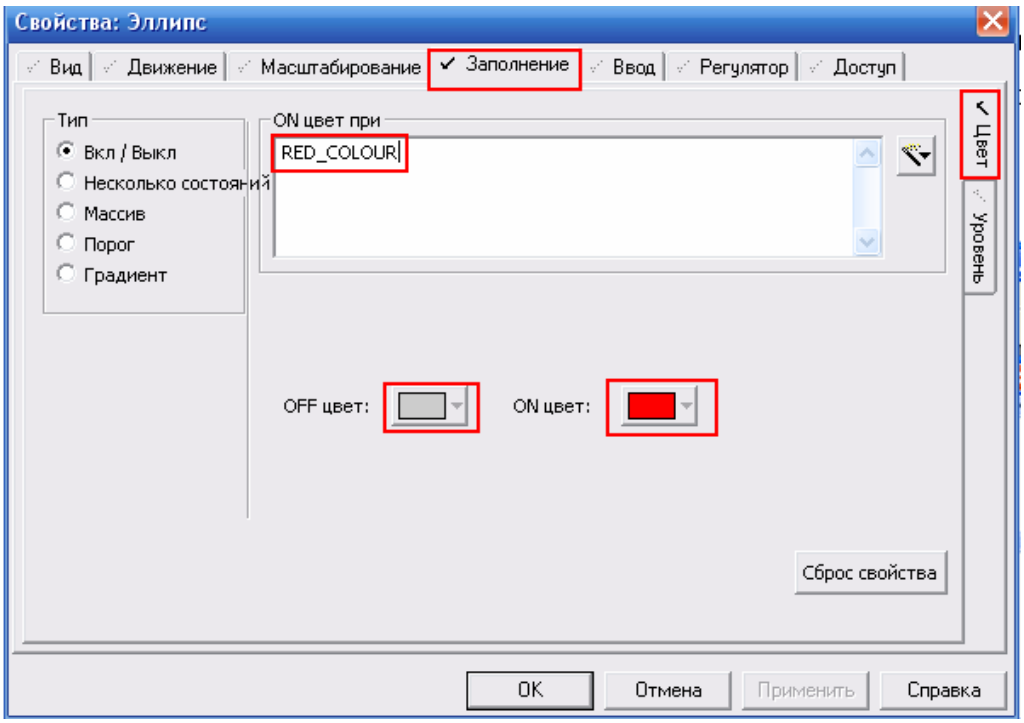



Рисунок 5.22 – Свойства объекта «Эллипс».

Так как окружностей нужно три, необходимо создать еще две. Можно снова выбрать пиктограмму «Эллипс» и проделать вышеописанные действия, либо кликнуть по созданному объекту правой клавишей мыши – копировать, снова кликнуть на странице правой клавишей мыши – вставить. Но только не забудьте войти в свойства объектов (рисунок 5.22), заменить во вкладке «Заполнение» тэг RED_COLOR на YELLOW_COLOUR и GREEN_COLOUR соответственно и изменить «ON цвет» на соответствующий.

Чтобы отображался таймер до включения зеленого цвета, создайте объект «Число» ниже корпуса светофора и свяжите его с тэгом (смотри рисунок 5.23).Создайте объект для задания времени цикла: пиктограмма «Число» вставьте в удобном месте. Свойства «Вид» аналогичны свойствам таймера, вкладка ввод заполняется в соответствии с рисунком 5.24.

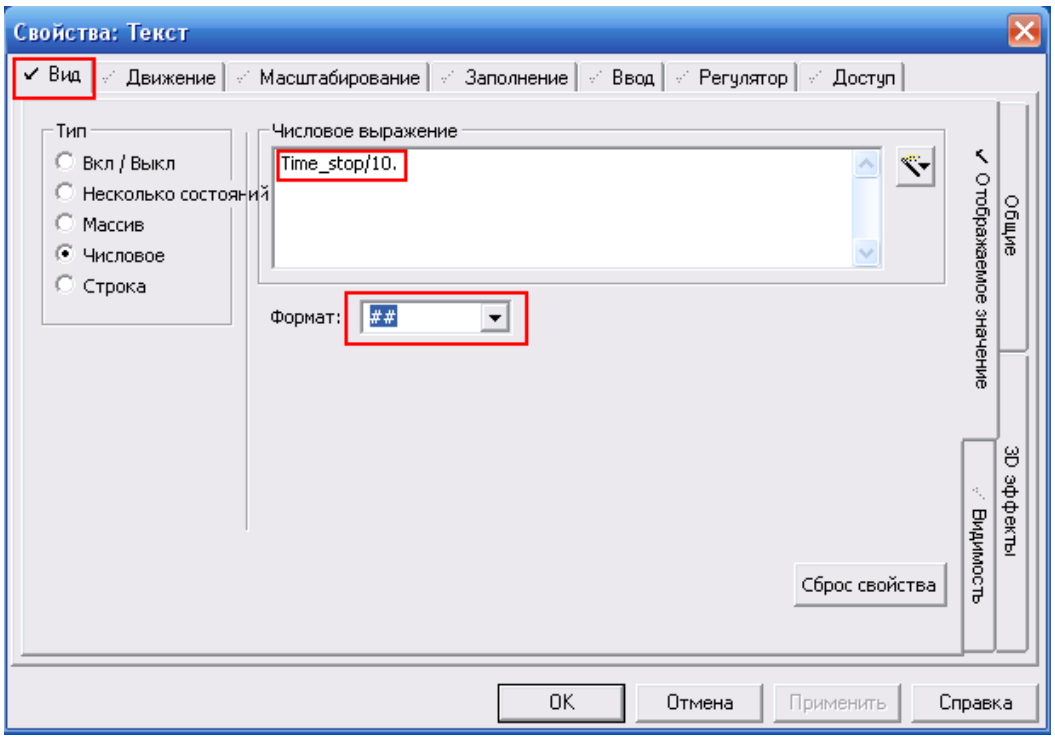



Рисунок 5.23 – Заполнение свойств таймера

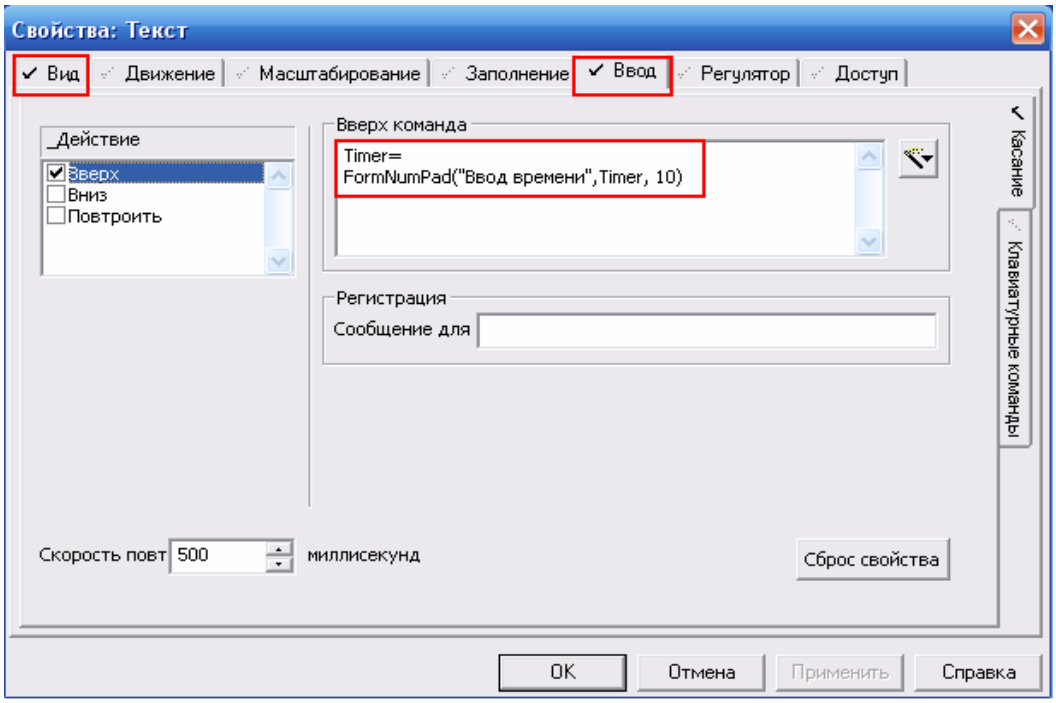



Рисунок 5.24 – Заполнение свойств объекта задания цикла.

После ввода всех тэгов и создания экранной формы выберите в «Редакторе проектов», в меню «Файл – Компилировать» (ALT+F10). Если при компиляции произошли ошибки, исправьте их. Затем скомпилируйте проект еще раз и, если компилирование прошло успешно, нажмите «Файл – Выполнить» (F5). После этого Citect запустит ваш проект на исполнение (смотри рисунок 5.25).

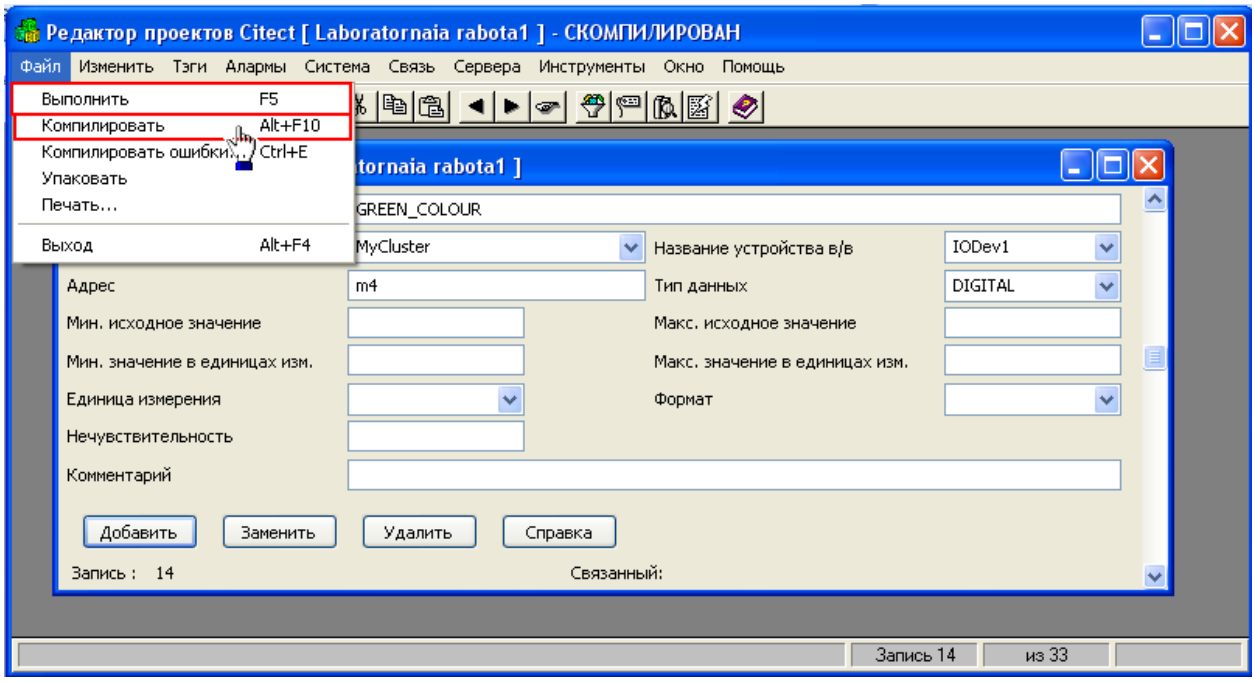



Рисунок 5.25 – Запуск проекта на исполнение

После запуска появится окно приветствие Citec, система предупредит, что работает в Демо-режиме, это20 минут. Затем проект достаточно просто перезапустить. Далее выберите меню «Обновить список страниц”, и затем вашу страницу – Page1.

Программа работает следующим образом. В поле время цикла вы вводите время цикла работы светофора в секундах. Щелкните по полю ввода, появится экранная клавиатура для ввода значения.

После этого, если вы включите x1 на панели стенда, то светофор заработает в цикле красный-желтый-зеленый, на табло внизу будет отображаться время до включения зеленого цвета. Включение x2 остановит цикл и включит постоянно красный цвет. Аналогично, x3 включит желтый, x4

– зеленый цвет. При использовании GX-Simulator переключение входов осуществляется манипуляциями в симуляторе.

Если все работает согласно описанию, то лабораторная работа завершена. Для реализации возможности восстановления своего проекта на другом ПК, используйте функцию резервного копирования проекта и его восстановления. Для создания резервной копии перейдите в окно «Проводник Citect»,на панели инструментов кликните иконку «Резервная копия» (рисунок 5.26). В меню выберите директорию для сохранения.

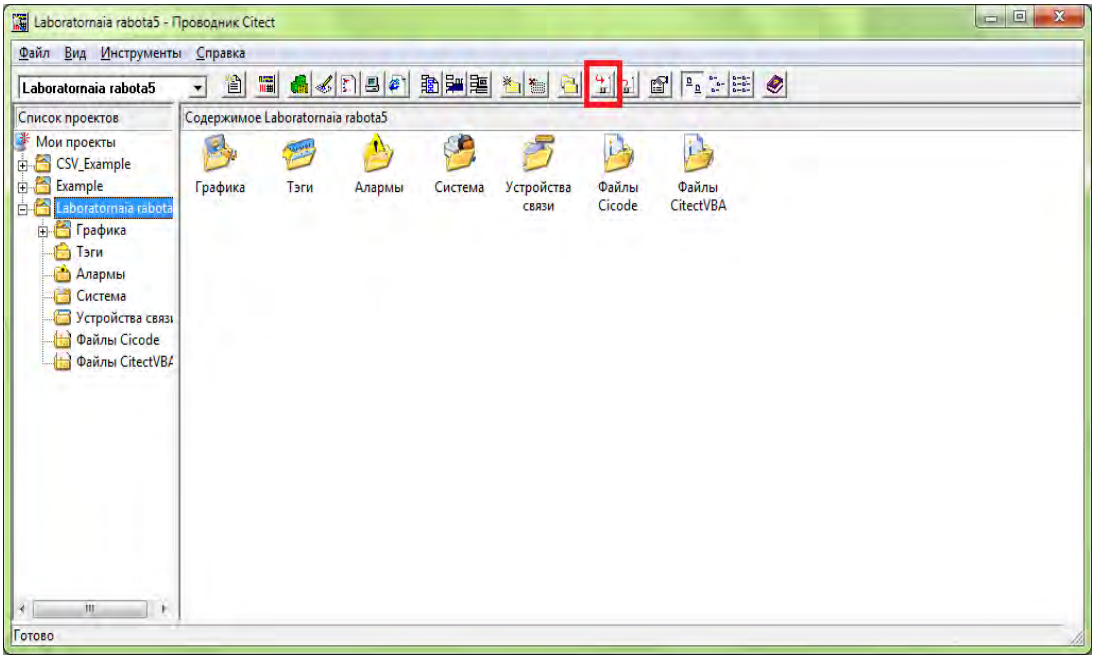



Рисунок 5.26 – Создание резервной копии проекта.

Для восстановления своего проекта на другом ПК, используйте «Восстановить» в том же окне «Проводник Citect» (рисунок 5.27). В меню указать директорию, где находится резервная копия проекта.

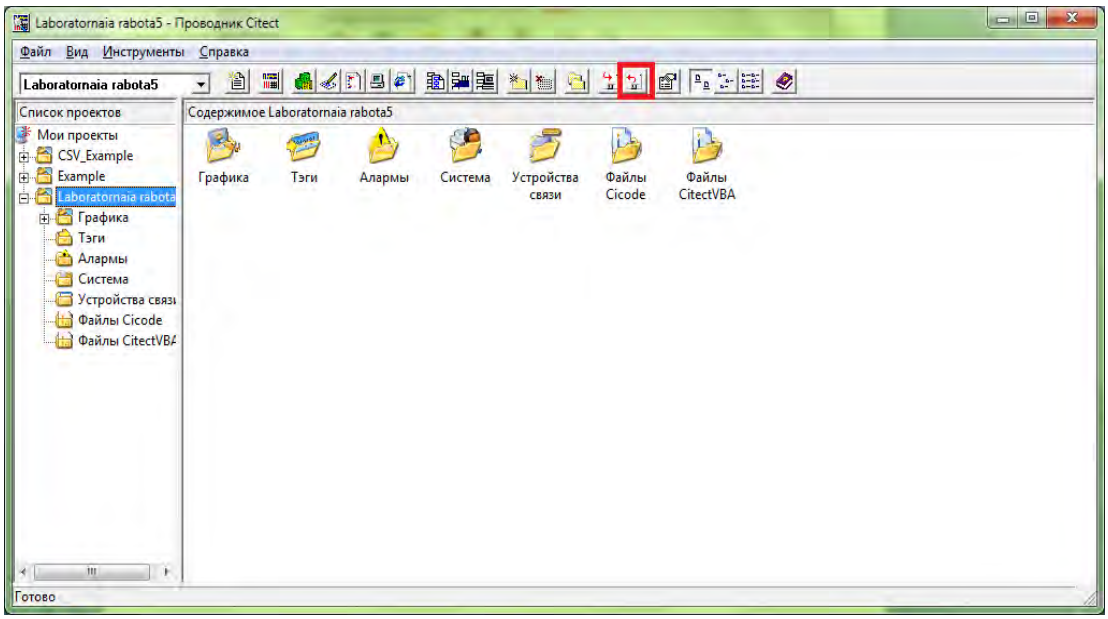



Рисунок 5.27 – Восстановление резервной копии.

**Контрольные вопросы.**

1. Перечислите этапы создания нового проекта в системе Citect.
2. Составьте алгоритм установления связи с устройством ввода-вывода.
3. Что такое сервер ввода-вывода (алармов, трендов)?
4. Что такое тэг? Как создается тэг?
5. Предложите свой вариант проекта для проверки связи с ПЛК.
6. Составьте алгоритм создания резервной копии проекта и его восстановления.
7. Каков порядок создания светофора в графическом редакторе?
8. Как осуществить проверку проекта на наличие ошибок и его запуск?

# III.КОНТРОЛЬ ЗНАНИЙ

## 1.Экзаменационные вопросы

1. Обобщенная структурная схема системы автоматического управления (САУ).
2. Классификация САУ.
3. Разновидности САУ.
4. Следящие САУ.
5. Принцип действия устройства числового программного управления (УЧПУ).
6. Классификация УЧПУ.
7. Цикловые, контурные, позиционные системы ЧПУ.
8. Сравнительная характеристика ЧПУ различных типов.
9. Обобщенная структурная схема УЧПУ.
10. Функциональные блоки УЧПУ.
11. Принципы программирования УЧПУ.
12. Адаптивные САУ.
13. Сравнительная характеристика САУ различных видов.
14. Система координат станка с ЧПУ.
15. Нулевые точки станка, детали и программы.
16. Принцип кодирования информации в управляющих программах УЧПУ.
17. Код ISO-7bit.
18. Структура управляющей программы (УП). Формат кадра УП УЧПУ.
19. Линейная интерполяция в УЧПУ.
20. Круговая интерполяция в УЧПУ.
21. Основные функции УП в УЧПУ.
22. Вспомогательные функции УП в УЧПУ.
23. Подготовительные функции УП в УЧПУ.
24. Назначение и классификация программируемых логических контроллеров (ПЛК).
25. Сравнительная характеристика ПЛК различных типов.
26. Структура ПЛК.
27. Устройство и принцип действия ПЛК.
28. Языки программирования ПЛК.
29. Запись логических функций на языке РКС.
30. Программирование логических элементов на языке РКС.
31. Типовые программные модули на языке РКС.
32. Инструкции процесса обработки программы ПЛК.
33. Типичные примеры применения ПЛК.
34.. Назначение и область применения микроконтроллеров (МК).
35. Устройство и принцип действия МК.
36. Структура арифметико-логического устройства МК.
37. Форматы команд МК.

38.Способы адресации в МК.
39. Основные группы команд МК.
40.Диспетчерское управление в АСУТП.
41. Назначение и функции SCADA-систем.
42. Основные термины и определения в SCADA-системах.
43.Программные средства для обеспечения и управления жизненным циклом изделия.

## IV.ТИПОВАЯ ПРОГРАММА ПО ДИСЦИПЛИНЕ

## ПРОГРАММНОЕ УПРАВЛЕНИЕ ТЕХНОЛОГИЧЕСКИМ ОБОРУДОВАНИЕМ

## ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Типовая учебная программа по учебной дисциплине «Программное управление технологическим оборудованием» разработана в соответствии с требованиями образовательного стандарта по направлению специальности 1-53 01 01-02 «Автоматизация технологических процессов и производств (в приборостроении и радиоэлектронике)».

Учебная дисциплина «Программное управление технологическим оборудованием» относится к циклу специальных дисциплин и является ключевой в подготовке инженера по автоматизации.

Цель преподавания учебной дисциплины – теоретическая и практическая подготовка студентов в области программирования автоматизированного технологического оборудования.

Задача преподавания учебной дисциплины – дать студентам представление о методах и языках программирования автоматизированного технологического оборудования, такого как станки, пресса, литейные и сварочные машины, сборочные автоматы и т.п.

В результате освоения учебной дисциплины студент должен

**знать:**

– полный объем задач программного управления;

– методы программирования систем управления станков, роботов, программируемых логических контроллеров;

– программирование систем управления отдельных объектов;

– общие принципы математического обеспечения и реализации задач числового программного управления (ЧПУ);

– особенности задач ЧПУ в связи с необходимостью работы в режиме реального времени;

**уметь:**

– разрабатывать технические задания на создание систем управления, выбирать технические и программные средства;

– программировать станки, роботы системы управления технологическим оборудованием;

– разрабатывать управляющие программы для микроконтроллеров, программируемых логических контроллеров;

– проводить подготовку производства с использованием систем автоматического программирования;

– использовать базы данных по оборудованию, инструментам и технологическим процессам;

– разрабатывать интерфейс и осуществлять программную интеграцию с периферией систем централизованного мониторинга, диспетчирования и визуализации производственного процесса.

Изучение данной учебной дисциплины базируется на следующих курсах: «Теория автоматического управления», «Электроника и микропроцессорная техника», «Электрические машины и автоматизированный электропривод».

**Методы (технологии) обучения**

Основными методами обучения, отвечающими целям изучения учебной дисциплины, являются:

– элементы проблемного обучения (проблемное изложение, вариативное изложение, частично-поисковый метод), реализуемые на лекционных занятиях;

– элементы учебно-исследовательской деятельности, реализуемые на лабораторных занятиях и при самостоятельной работе;

– коммуникативные технологии (дискуссия, учебные дебаты, «мозговой штурм» и другие формы и методы), реализуемые на практических занятиях;

– проектные технологии, используемые при проектировании конкретного объекта, реализуемые при выполнении курсовой работы.

**Организация самостоятельной работы студентов**

При изучении учебной дисциплины рекомендуется использовать следующие формы самостоятельной работы:

– контролируемая самостоятельная работа в виде решения индивидуальных задач в аудитории во время проведения практических занятий под контролем преподавателя в соответствии с расписанием;

– управляемая самостоятельная работа, в том числе в виде выполнения индивидуальных расчетных заданий с консультациями преподавателя;

– подготовка рефератов по индивидуальным темам, в том числе с использованием патентных материалов;

– подготовка курсовой работы по индивидуальным заданиям, в том числе разноуровневым заданиям.

Согласно типовому учебному плану на изучение учебной дисциплины «Программное управление технологическим оборудованием» отведено 310 часов, в том числе 176 часов аудиторных занятий, из них лекции – 64 часа, лабораторные занятия – 64 часа, практические занятия – 48 часов.

**Примерный тематический план**

| Наименование раздела и темы | Количество аудиторных часов | | | |
|---|---|---|---|---|
| | **Лекции** | **Практические занятия** | **Лабораторные занятия** | **Всего** |
| **Раздел I. Системы автоматического управления (САУ)** | 8 | | | 8 |
| Тема 1.1. Общие сведения о системах автоматического управления технологическим оборудованием | 2 | - | | 2 |
| Тема 1.2. Разновидности САУ | 6 | | | 6 |
| **Раздел II. Программирование систем управления** | 46 | 48 | 48 | 142 |
| Тема 2.1. Программирование программируемых логических контроллеров (ПЛК) | 12 | 16 | 8 | 36 |
| Тема 2.2.Программирование микроконтроллеров (МК) | 12 | 16 | 8 | 36 |
| Тема 2.3. Программирование систем числового программного управления (ЧПУ) | 22 | 16 | 32 | 70 |
| **Раздел III. Программные средства интеграции** | 10 | | 16 | 26 |
| Тема 3.1. Средства программирования систем централизованного мониторинга, управления, диспетчирования и визуализации производственных процессов (SCADA-системы) | 8 | | 16 | 24 |

| | | | | |
|---|---|---|---|---|
| Тема3.2. Программные средства для обеспечения и управления жизненного цикла изделия (CALS-технологии и PLM-системы) | 2 | | | 2 |
| **ВСЕГО** | 64 | 48 | 64 | 176 |

## СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

### Раздел I. Системы автоматического управления (САУ)

### Тема 1.1 Общие сведения о системах автоматического управления технологическим оборудованием

Обобщенная структурная схема САУ. Классификация САУ.

### Тема 1.2. Разновидности САУ

Механические копировальные системы. Цикловое программное управление (ЦПУ). Следящие САУ. Системы числового программного управления (ЧПУ): принцип действия, достоинства, классификация, обобщенная структурная схема, функциональные блоки, принципы программирования, структура управляющей программы (УП). Адаптивные САУ. Сравнительная характеристика САУ различных видов.

### Раздел II. Программирование систем управления

### Тема 2.1. Программирование программируемых логических контроллеров (ПЛК)

Назначение ПЛК. Предшественники и объективная необходимость появления. Устройство и принцип действия. Типичные примеры применения. Представители и их сравнительные характеристики.

Языки программирования: релейно-контактных схем (LD), функциональных блоков (FBD), последовательности функциональных схем (SFC), списка инструкций (IL), структурированного текста (ST).

Структура LD-программы. Типовые программные модули LD.

### Тема 2.2. Программирование микроконтроллеров (МК)

Назначение МК. Предшественники и объективная необходимость появления. Устройство и принцип действия. Типичные примеры применения. Представители, структура и их сравнительные характеристики. Язык ассемблера для программирования МК. Разработка УП МК на языке ассемблера. Основные сведения о языке С. Разработка УП МК на языке С. Примеры УП МК для управления технологическим оборудованием.

### Тема 2.3. Программирование систем числового программного управления (ЧПУ)

Технические средства устройств ЧПУ (УЧПУ): пульты оператора, процессоры, электроавтоматика, привода. Система координат станка с ЧПУ. Нулевые точки станка, детали и программы. Принцип кодирования информации в управляющих программах. Код ISO-7bit. Структура УП. Формат кадра УП. Символы кода. Пример УП. Вспомогательные функции и

стандартные циклы в УП. Подготовительные функции в УП. Типовые программные конструкции УП.

## Раздел III Программные средства интеграции

### Тема 3.1. Средства программирования систем централизованного мониторинга, управления, диспетчирования и визуализации производственных процессов (SCADA-системы)

Программные средства автоматизации, связь с физическими устройствами. Назначение и функции SCADA-систем. Основные термины и определения. Свойства SCADA-систем. Типичные примеры применения. Представители и их сравнительные характеристики. Примеры разработки SCADA-системы для различного автоматизированного оборудования.

### Тема 3.2. Программные средства для обеспечения и управления жизненного цикла изделия (CALS-технологии и PLM-системы)

Назначение и типичные примеры использования. Функции программного обеспечения. Примеры программной реализации некоторых функций.

## ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

### Примерный перечень тем практических занятий

1. Выбор типа и структуры ПЛК в зависимости от назначения системы управления.
2. Разбор примера УП ПЛК на языке РКС.
3. Программирования ПЛК на обучающей программе:
   - программирование логических функций;
   - программирование функциональных модулей.
4. Разработка УП ПЛК для циклового ПР.
5. Разработка УП ПЛК для робототехнического комплекса.
6. Разработка УП ПЛК для управления конвейером.
7. Разбор программы ПЛК на языке STL.
8. Выбор типа и структуры МК в зависимости от назначения системы управления.
9. Разбор примера УП микроконтроллера на языке ассемблера.
10. Разбор примера УП микроконтроллера на языке СИ.
11. Разработка УП микроконтроллера на языке СИ для управления электродвигателем:
    - реверсирование без регулирования оборотов;
    - регулирования оборотов с помощью программной ШИМ;
    - регулирования оборотов с помощью аппаратной ШИМ.
12. Разработка УП микроконтроллера с использованием таймера.
13. Сравнительный анализ возможности использования ПЛК и МК в системах управления различных типов.
14. Сравнительный анализ различных систем ЧПУ.
15. Анализ структуры УЧПУ типа CNC.
16. Формат кадра УП ЧПУ.
17. Разбор примера УП ЧПУ для обработки детали типа тела вращения.
18. Разработка УП ЧПУ для обработки детали типа тела вращения:
    -УЧПУ типа NC;
    -УЧПУ типа CNC.
19. Разбор примера УП ЧПУ для обработки детали типа корпус.
20. Разработка УП ЧПУ для обработки детали типа корпус.

### Примерный перечень тем лабораторных занятий

1. Отладка УП ПЛК для стенда автоматизированного управления светофорами перекрестка.
2. Отладка УП ПЛК для стенда автоматизированного рекламного щита.
3. Отладка УП ПЛК для стенда автоматизированного привода.
4. Отладка УП ПЛК для циклового промышленного робота.
5. Отладка программ для микроконтроллера на языке СИ:

-включение двигателя и светодиода (индикатора) на определенный промежуток времени;

-использования программного таймер для включения двигателя на определенный промежуток времени;

-программная широтно-импульсная модуляция (ШИМ) для управления мощностью двигателя.

6. Отладка управляющей программы ЧПУ обработки детали типа тела вращения на симуляторе.
7. Отладка управляющей программы ЧПУ обработки детали типа корпус на симуляторе.
8. Управление УЧПУ с пульта оператора.
9. Ввод УП в УЧПУ.
10. Взаимодействие основных частей УЧПУ на примере реальной стойке ЧПУ и макете исполнительного станочного механизма.
11. Отладка управляющих программ обработки деталей различных типов на реальной стойке ЧПУ и макете исполнительного станочного механизма .
12. Диагностика УЧПУ на примере реальной стойке ЧПУ и макете исполнительного станочного механизма .
13. Отладка программ взаимодействия в среде Citect ПЛК и SCADA-системы:

-управления парковкой;

-управления наполнением трех емкостей;

-управления работой двух конвейеров;

-управление работой роботизированного пресса.

## Примерное содержание курсовой работы

Тема: Разработка программного обеспечения для управления робототехническим комплексом.

Цель: разработка программной части системы управления РТК.

Исходные данные: чертеж детали-представителя, схема технологического процесса, структура робототехнического комплекса. Содержание: разработка управляющей программы для станка с ЧПУ; разработка пульта управления робототехнического комплекса, разработка программы ПЛК для управления основным и периферийным оборудованием; разработка интерфейса экранного пульта SCADA-системы и эмулятора работы РТК на ее основе.

Графический материал должен иллюстрировать все этапы и результаты разработки.

Примерный объем работы: 30 листов, включая схемы, алгоритмы и листинги программ. Примерное число часов на выполнение: 60.

## Список литературы


**Основная литература**

1. Сосонкин, В.Л. Методика программирования станков с ЧПУ на наиболее полном полигоне вспомогательных G-функций / В.Л. Сосонкин, Г.М. Мартинов – М.: Мосстанкин, 2007. – 231стр.
2. Методическое пособие по программированию логических программируемых контроллеров фирмы Мицубиси / Ю.Е. Лившиц [и др.]; под общ. ред. Ю.Е. Лившица – Минск: БНТУ, 2010. – 204стр.
3. Евстифеев, А.В. Микроконтроллеры AVR семейства Mega / А.В. Евстифеев–М.: Издательский дом «ДОДЭКАХХІ» – 2007. – 592стр.

**Дополнительная литература**

4. Денисенко, В.В. Компьютерное управление оборудованием, технологическим процессом, экспериментом / В.В. Денисенко – .М.:Телеком, 2009. – 608стр.
5. Бергер, Г. Инструкция по программированию логических программируемых контроллеров фирмы Сименс / Г.Бергер– М.: Имприс, 2008. – 723стр.
6. Олсон, Г., Пиани Д. Цифровые системы автоматизации и управления / Г.Олсон., Д.Пиани – СПб.: Невский Диалект, 2001. – 476стр.
7. Техническое описание УЧПУ «ИРИС М64», Минск 2008.


## Диагностика компетенций студента

Для оценки достижений студента рекомендуется использовать следующий диагностический инструментарий:

– устный и письменный опрос во время лабораторных и практических занятий;
– защита выполненных лабораторных работ;
– защита курсовой работы;
– сдача зачета;
– сдача экзамена.

## Критерии оценки результатов учебной деятельности

| Баллы | Критерии оценки |
|---|---|
| 1 **(один)** | Отсутствие приращения знаний и компетентности в рамках дисциплины; отказ от ответа |
| 2 **(два)** | Фрагментарные знания в рамках дисциплины; знание отдельных литературных источников, рекомендованных учебной программой дисциплины; неумение использовать научную терминологию дисциплины, наличие в ответе грубых ошибок; пассивность на практических и лабораторных занятиях, низкий уровень культуры исполнения заданий |
| 3 **(три)** | Недостаточно полный объем знаний в рамках дисциплины; знание части основной литературы, рекомендованной учебной программой дисциплины; использование научной терминологии, изложение ответа на вопросы с существенными ошибками; слабое владение инструментарием учебной дисциплины, неумение ориентироваться в основных теориях, методах и направлениях дисциплины; пассивность на практических и лабораторных занятиях; низкий уровень культуры исполнения заданий |
| 4 **(четыре)** | Достаточный объем знаний в рамках дисциплины; усвоение основной литературы, рекомендованной учебной программой дисциплины; использование научной терминологии, логическое изложение ответа на вопросы, умение делать выводы без существенных ошибок; владение инструментарием учебной дисциплины, умение под руководством преподавателя решать стандартные (типовые) задачи; умение ориентироваться в основных теориях, методах и направлениях дисциплины и давать им оценку; работа под руководством преподавателя на практических и лабораторных занятиях, допустимый уровень культуры исполнения заданий |
| 5 **(пять)** | Достаточные знания в объеме учебной программы; использование научной терминологии, грамотное, логически правильное изложение ответа на вопросы, умение делать выводы; владение инструментарием учебной дисциплины, умение его использовать в решении учебных задач; способность самостоятельно применять типовые решения в рамках учебной программы; усвоение основной литературы, рекомендованной учебной программой дисциплины; умение ориентироваться в теориях, методах и направлениях дисциплины и давать им сравнительную оценку; самостоятельная работа на практических и лабораторных занятиях, фрагментарное участие в групповых обсуждениях, достаточный уровень культуры исполнения заданий |

| | |
|---|---|
| 6 **(шесть)** | Достаточно полные и систематизированные знания в объеме учебной программы; использование необходимой научной терминологии, грамотное, логически правильное изложение ответа на вопросы, умение делать обобщения и обоснованные выводы; владение инструментарием учебной дисциплины, умение его использовать в решении учебных задач; способность самостоятельно применять типовые решения в рамках учебной программы; усвоение основной литературы, рекомендованной учебной программой дисциплины; умение ориентироваться в теориях, методах и направлениях дисциплины и давать им сравнительную оценку; самостоятельная работа на практических и лабораторных занятиях, периодическое участие в групповых обсуждениях, достаточно высокий уровень культуры исполнения заданий |
| 7 **(семь)** | Систематизированные, глубокие и полные знания по всем разделам учебной программы; использование научной терминологии, грамотное, логически правильное изложение ответа на вопросы, умение делать обоснованные выводы и обобщения; владение инструментарием учебной дисциплины, умение его использовать в постановке и решении научных задач; свободное владение типовыми решениями в рамках учебной программы; усвоение основной и дополнительной литературы, рекомендованной учебной программой дисциплины; умение ориентироваться в основных теориях, методах и направлениях дисциплины и давать им аналитическую оценку; активная самостоятельная работа на практических и лабораторных занятиях, участие в групповых обсуждениях, высокий уровень культуры исполнения заданий |
| 8 **(восемь)** | Систематизированные, глубокие и полные знания по всем поставленным вопросам в объеме учебной программы; использование научной терминологии, грамотное и логически правильное изложение ответа на вопросы, умение делать обоснованные выводы и обобщения; владение инструментарием учебной дисциплины, умение его использовать в постановке и решении научных задач; способность самостоятельно решать сложные проблемы в рамках учебной программы; усвоение основной и дополнительной литературы, рекомендованной учебной программой дисциплины; умение ориентироваться в теориях, методах и направлениях дисциплины и давать им аналитическую оценку; активная самостоятельная работа на практических и лабораторных занятиях, систематическое участие в групповых обсуждениях, высокий уровень культуры исполнения заданий |

| | |
|---|---|
| 9 **(девять)** | Систематизированные, глубокие и полные знания по всем разделам учебной программы; точное использование научной терминологии, грамотное, логически правильное изложение ответа на вопросы; владение инструментарием учебной дисциплины, умение его эффективно использовать в постановке и решении научных задач; способность самостоятельно и творчески решать сложные проблемы в нестандартной ситуации в рамках учебной программы; полное усвоение основной и дополнительной литературы, рекомендованной учебной программой дисциплины; умение ориентироваться в теориях, методах и направлениях дисциплины и давать им аналитическую оценку; систематическая активная самостоятельная работа на практических и лабораторных занятиях, творческое участие в групповых обсуждениях, высокий уровень культуры исполнения заданий |
| 10 **(десять)** | Систематизированные, глубокие и полные знания по всем разделам учебной программы, а также по основным вопросам, выходящим за ее пределы; точное использование научной терминологии, грамотное, логически правильное изложение ответа на вопросы; безупречное владение инструментарием учебной дисциплины, умение его эффективно использовать в постановке и решении научных задач; выраженная способность самостоятельно и творчески решать сложные проблемы в нестандартной ситуации; полное и глубокое усвоение основной и дополнительной литературы по учебной дисциплине; умение свободно ориентироваться в теориях, методах и направлениях дисциплины и давать им аналитическую оценку, использовать научные достижения других дисциплин; самостоятельная творческая работа на практических и лабораторных занятиях, активное творческое участие в групповых обсуждениях, высокий уровень культуры исполнения заданий. |

**Примерный перечень компьютерных программ**

- Симуляторы программных перемещений станков с ЧПУ.
- Обучающая программа и симулятор ПЛК.
- Редактор программ ПЛК.
- Компьютерный программатор МК.
- Программная среда Citect.
- Программная среда Proteus.
- Программная среда **CodeVisionAVR.**

## СПИСОК ЛИТЕРАТУРЫ


1. Сосонкин, В.Л. Методика программирования станков с ЧПУ на наиболее полном полигоне вспомогательных G-функций / В.Л. Сосонкин, Г.М. Мартинов – М.: Мосстанкин, 2007. – 231стр.
2. Методическое пособие по программированию логических программируемых контроллеров фирмы Мицубиси / Ю.Е. Лившиц [и др.]; под общ. ред. Ю.Е. Лившица ч1 – Минск: БНТУ, 2010. – 204стр.
3. Методическое пособие по программированию логических программируемых контроллеров фирмы Мицубиси / Ю.Е. Лившиц [и др.]; под общ. ред. Ю.Е. Лившица ч2 – Минск: БНТУ, 2014. – 195стр.
4. Программирование на языке С для AVR и PIC микроконтроллеров / Сост. Ю.А. Шпак-К.:"МК-Пресс",2006.-400с.
5. SCADA—системы: взгляд изнутри/ Е.Б. Андреев и др.-М.: Издательство "РТСсофт", 2004-176с.
6. Программирование микроконтроллеров/ лабораторный практикум для студентов специальности 1-53 01 01/сост.: Ф.Л. Сиротин [и др.-Минск.БНТУ,2014-Ч1.-2014.-64с.
7. Руководство по выполнению лабораторных работ на стенде НТЦ-02.31.2 "Микропроцессорная техника PIC "-Могилев.-162с.
8. Современный станок с ЧПУ и CAD/CAM система.-А.А. Ловыгин [и др.]-М.:"ЭльфИПР", 2006, 286с.