МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Электропривод и автоматизация промышленных установок»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МЕХАТРОННЫХ СИСТЕМ

Методические рекомендации к практическим занятиям для студентов направления подготовки 15.04.06 «Мехатроника и робототехника» очной и заочной форм обучения

Могилев 2023

УДК 621.865:004.45
ББК 3.816:32.9732
П78

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Электропривод и автоматизация промышленных установок» «2» мая 2023 г., протокол № 7

Составитель О. А. Капитонов

Рецензент канд. техн. наук, доц. Е. В. Ильюшина

Методические рекомендации к практическим занятиям предназначены для студентов направления подготовки 15.04.06 «Мехатроника и робототехника» очной и заочной форм обучения.

Учебное издание

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МЕХАТРОННЫХ СИСТЕМ

Ответственный за выпуск	А. С. Коваль
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

© Белорусско-Российский
университет, 2023

Содержание

1 Практическая работа № 1. Методы оценки качества программного обеспечения мехатронных систем

Цель работы:

– освоение терминологии в области оценки качества и жизненного цикла программного обеспечения (ПО);

– изучение процессов жизненного цикла ПО мехатронных систем (МС);

– изучение методов оценки качества программного обеспечения мехатронных систем;

– практическое рассмотрение нормативных документов в сфере оценки качества программного обеспечения;

– получение практических навыков оценки качества программного обеспечения мехатронных систем.

1.1 Задание

Заданием к работе является изучение нормативных документов по оценке качества ПО и определение показателей качества программного обеспечения, на примере заданной преподавателем прикладной программы (программного средства). Обучающийся получает у ведущего работу преподавателя номер варианта, согласно которого выбираются определенные виды показателей качества ПО в соответствии с методикой, изложенной в указанном нормативном документе. Обучающийся должен выполнить следующее:

1) изучить процессы жизненного цикла программного средства (прикладной программы или программной системы);
2) изучить указанный нормативный документ по оценке качества ПО;
3) установить критерии оценки качества (метрики) программного средства (прикладной программы);
4) оценить качество устойчивости функционирования заданной программы;
5) оценить качество обработки ошибочных (исключительных) ситуаций;
6) оценить качество восстановления процесса выполнения программы после возникновения аппаратного или программного сбоя;
7) оценить качество восстановления результатов расчета по программе после возникновения аппаратного или программного сбоя;
8) выполнить диагностику граничных и аварийных ситуаций с программой;
9) определить требования к динамическому тестированию программы;
10) определить требования к статическому тестированию программы;
11) выполнить оценку качества документации, имеющейся на программное обеспечение.

1.2 Ход выполнения работы

1.2.1 Изучение терминологии в области качества и жизненного цикла программного обеспечения.

В начале работы следует изучить терминологию в области жизненного цикла программного обеспечения и оценки качества программных средств, прочитав раздел 3, в представленных на персональном компьютере (ПК) или компьютерной сети электронных версии заданных нормативных документов.

1.2.2 Изучение структуры и процессов жизненного цикла программных средств.

Следует рассмотреть указанную в СТБ ИСО/МЭК 12207–2003 структуру и состав процессов жизненного цикла. Следует отметить в отчете состав основных, вспомогательных и организационных процессов жизненного цикла программных средств (прикладного программного обеспечения МС).

1.2.3 Выбор критериев оценки качества (*шкал и метрик*) *ПО.*

Вначале согласно п. 2.2 ГОСТ 28195 определить к какому классу ПО относится заданная программа. В зависимости от класса ПО согласно таблице 2 из ГОСТ 28195 определяются показатели, оказывающие влияние качество программы.

Далее выбирается тип шкал, которые будут использоваться при оценке качества заданной программы: номинальная, порядкового типа, интервального типа или относительная. Два первых типа шкал следует применять для оценки качественных атрибутов заданной программы, которые нельзя измерить количественно, а также для ранжирования измеренных значение, третий и четвертый типы шкал – использовать для оценки количественных атрибутов качества программы. Для показателей качества на всех уровнях принимается единая шкала оценки от 0 до 1.

Расчет дальнейших показателей качества следует оформить в виде таблицы, используя табличный или текстовый редактор.

1.2.4 Оценка качества устойчивости функционирования программы.

Сначала определяется устойчивость функционирования программы при наличии ошибок во входных данных – показатель H0101. При этом выполняется качество контроля корректности входных данных; наличие контроля принадлежности входных данных к диапазону допустимых значений; наличие контроля форматов входных данных; наличие диагностических сообщений об ошибке пользователя и предпринимаемые действия, связанные с обработкой возникшей ситуации при вводе ошибочных данных.

1.2.5 Оценка качества обработки программой ошибочных (*исключительных*) *ситуаций.*

Вначале следует составить список возможных исключительных ситуаций, возникновение которых возможно в рассматриваемой системе, и определить

желаемую реакцию системы на возникновение данных ситуаций. Затем производится имитация ситуации, приводящей к возникновению исключительной ситуации и изучается поведение ПО и системы управления. Делается вывод о соответствии наблюдаемого поведения системы определенному ранее желаемому поведению.

1.2.6 Оценка качества восстановления процесса выполнения программы после возникновения аппаратного или программного сбоя.

Необходимо составить перечень возможных аппаратных и программных сбоев, которые могут возникнуть в рассматриваемой системе. Затем следует имитировать ситуации, в которых возникают данные виды сбоев, задокументировать поведение системы при возникновении заданных видов сбоев.

1.2.7 Оценка качества восстановления результатов расчета по программе после возникновения аппаратного или программного сбоя.

Необходимо составить перечень возможных аппаратных и программных сбоев, возникновение которых предполагает необходимость восстановления результатов расчета и продолжения работы системы. Затем следует имитировать ситуации, в которых возникают данные виды сбоев, задокументировать поведение системы при возникновении заданных видов сбоев, отметить, насколько полно происходит восстановление результатов расчета после сбоя.

1.2.8 Разработка процедуры динамического тестирования программы.

Процесс и функции динамического тестирования при разработке программного обеспечения динамическое тестирование можно разделить на модульное тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование и, наконец, регрессионное тестирование.

Модульное тестирование – это тест, который фокусируется на корректности основных компонентов программного обеспечения. Модульное тестирование относится к категории тестирования «белого ящика». Во всей системе контроля качества модульное тестирование должно быть завершено группой продуктов, а затем программное обеспечение передается в отдел тестирования.

Интеграционное тестирование используется для определения того, правильно ли подключены интерфейсы между различными блоками в процессе интеграции всего программного обеспечения.

Тестирование программной системы, завершившей интеграцию, называется системным тестированием, и целью теста является проверка того, что корректность и производительность программной системы соответствуют требованиям, указанным в ее спецификациях. Тестировщики должны следовать установленному плану тестирования. При тестировании надежности и простоты использования программного обеспечения его входные и выходные данные и другие динамические эксплуатационные характеристики следует сравнивать со спецификациями программного обеспечения. Если программное обеспечение спецификация является неполной, тест-системы в большей степени зависит от

тестера опыт работы и решение такого теста недостаточно. Система тестирования «черного ящика» – тестирование.

Это заключительное тестирование перед вводом программного обеспечения в эксплуатацию. Это процесс тестирования программного обеспечения покупателем. В реальной работе компании оно обычно реализуется путем обращения к заказчику с просьбой опробовать или выпустить бета-версию программного обеспечения. Приемочный тест представляет собой тестирование «черного ящика».

Целью регрессионного тестирования является проверка и модификация результатов приемочных испытаний на этапе обслуживания программного обеспечения. В практических приложениях обработка жалоб клиентов является воплощением регрессионного тестирования.

1.2.9 Завершения работы с ПК. Формирование отчета.

В среде текстового редактора создается электронный документ отчета по выполненной работе. По окончанию выполнения работы с ПК выполняется резервное копирования полученных данных и отчета на внешний мобильный носитель или компьютерную сеть. Перед завершением сеанса работы с ПК следует с помощью файлового менеджера удалить ненужные файлы. По окончании работы выполняется выход из аккаунта учебной группы или выключение ПК.

1.3 Содержание отчета

Отчет по работе № 1 оформляется индивидуально каждым обучающимся на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе.

Состав отчета по выполненной практической работы следующий:

– титульный лист;
– текст индивидуального задания;
– разработанная шкала (метрики) критериев оценки качества ПО;
– оценка качества устойчивости функционирования заданной программы;
– оценка качества обработки ошибочных (исключительных) ситуаций;
– оценка качества восстановления процесса выполнения программы после возникновения аппаратного или программного сбоя;
– оценка качества восстановления результатов расчета по программе после возникновения аппаратного или программного сбоя;
– результаты диагностики граничных и аварийных ситуаций с программой;
– список требований к динамическому тестированию программы;
– список требований к статическому тестированию программы;
– оценка качества документации, имеющейся на заданное программное обеспечение.

Контрольные вопросы

1 Дать определения понятию «Жизненный цикл программных средств (программного обеспечения)».

2 Дать определения понятию «Качество».

3 Какие составляющие имеет жизненный цикл ПО МС?

4 Какую структуру имеет модель качества ПО?

5 Дать определение понятию «Функциональность (Functionality)» и перечислить ее составляющие.

6 Дать определение понятию «Надежность (Reliability)» и перечислить ее составляющие.

7 Дать определение понятию «Эффективность (Efficiency)» и перечислить ее составляющие.

8 Дать определение понятию «Сопровождаемость (Maintainability)» и перечислить ее составляющие.

9 Дать определение понятию «Мобильность (Portability» и перечислить ее составляющие.

10 Какие критерии предъявляются к оценке документации на программное обеспечение?

2 Практическая работа № 2. Формирование требований к ПО МС

Цель работы:

– изучение методов формирования требований к определенному виду программного обеспечения мехатронных систем;

– получение практических навыков формирования требований к определенному виду программного обеспечения мехатронных систем.

2.1 Задание

Составить перечень требований к программному обеспечению мехатронной системы, реализующему задачу управления промышленной установкой по заданию преподавателя.

2.2 Ход выполнения работы

2.2.1 Составить перечень требований функционального характера.

2.2.2 Составить перечень пользовательских требований.

2.2.3 Требования к показателям назначения (производительность, устойчивость к сбоям и т. п.).

2.2.4 Требования к эксплуатации и персоналу.

2.3 Теоретические сведения

Управление требованиями к программному обеспечению (англ. software requirements management) – процесс, включающий идентификацию, выявление, документирование, анализ, отслеживание, приоритезацию требований, достижение соглашения по требованиям и затем управление изменениями и уведомление соответствующих заинтересованных лиц. Управление требованиями – непрерывный процесс на протяжении всего проекта разработки программного обеспечения.

Цель управления требованиями состоит в том, чтобы гарантировать, что организация документирует, проверяет и удовлетворяет потребности и ожидания её клиентов и внутренних или внешних заинтересованных лиц. Управление требованиями начинается с выявления и анализа целей и ограничений клиента. Управление требованиями, далее, включает поддержку требований, интеграцию требований и организацию работы с требованиями и сопутствующей информацией, поставляющейся вместе с требованиями.

Установленная таким образом отслеживаемость требований используется для того, чтобы уведомлять заинтересованных участников об их выполнении, с точки зрения их соответствия, законченности, охвата и последовательности. Отслеживаемость также поддерживает управление изменениями как часть управления требованиями, так как она способствует пониманию того, как изменения воздействуют на требования или связанные с ними элементы, и облегчает внесение этих изменений.

Управление требованиями включает общение между проектной командой и заинтересованными лицами с целью корректировки требований на протяжении всего проекта. Постоянное общение всех участников проекта важно для того, чтобы ни один класс требований не доминировал над другими. Например, при разработке программного обеспечения для внутреннего использования у бизнеса могут быть столь сильные потребности, что он может проигнорировать требования пользователей, или полагать, что созданные сценарии использования покроют также и пользовательские требования.

Виды требований по уровням.

Бизнес-требования – определяют назначение ПО, описываются в документе о видении (vision) и границах проекта (scope).

Пользовательские требования – определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз-утверждений, в виде сценариев использования (англ. use case), пользовательских историй (англ. user stories), сценариев взаимодействия (scenario).

Функциональные требования – определяют требования к функциям системы.

Виды требований по характеру.

Бизнес-требования.

Пользовательские требования.

Функциональные требования.

Бизнес-правила – определяют ограничения, проистекающие из предметной области и свойств автоматизируемого объекта (предприятия).

Системные требования и ограничения – определения элементарных операций, которые должна иметь система, а также различных условий, которым она может удовлетворять.

Требования к документированию.

Требования к дизайну и юзабилити.

Требования к безопасности и надёжности.

Требования к показателям назначения (производительность, устойчивость к сбоям и т. п.).

Требования к эксплуатации и персоналу.

Прочие требования и ограничения (внешние воздействия, мобильность, автономность и т. п.).

2.4 Содержание отчета

Отчеты по работе № 2 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе в следующем составе:

– титульный лист;

– текст индивидуального задания;

– перечни требований, составленные по подразделу 2.3.

Контрольные вопросы

1 Какие требования относятся к бизнес-требованиям?

2 Какие требования относятся к пользовательским требованиям?

3 Какие требования относятся к функциональным требованиям?

4 Дайте определение понятию «Требования к программному обеспечению».

5 На какой стадии разработки формируется перечень требований к программному обеспечению?

3 Практическая работа № 3. Алгоритмическое обеспечение для ПО МС

Цель работы:

– изучение видов, способов описания и методов формирования алгоритмического обеспечения ПО МС;

– получение практических навыков создания алгоритмического обеспечения для ПО МС.

3.1 Задание

По заданию преподавателя, каждый студент должен получить модель асинхронного электродвигателя и преобразователя частоты, которые будут являться объектом управления в разрабатываемой системе. Используя паспортные данные, предоставляемые производителями выбранных моделей электродвигателя и преобразователя, выполнить следующие расчеты:

3.1.1 Рассчитать параметры схемы замещения асинхронного электродвигателя.

3.1.2 Рассчитать механическую и электромеханическую характеристики асинхронного электродвигателя.

3.1.3 Рассчитать искусственные механическую и электромеханическую характеристики, при регулировании различными способами.

3.1.4 Рассчитать передаточную функцию электродвигателя.

3.1.5 Рассчитать передаточную функцию преобразователя частоты.

3.1.6 Разработать алгоритмическую реализацию регуляторов в системе управления электроприводом, на основе выполненных ранее расчетов.

3.2 Теоретические сведения

Асинхронные двигатели с короткозамкнутым ротором находят широкое применение в приводах роботов и манипуляторов благодаря простоте их конструкции, надежности в эксплуатации, высоким энергетическим показателям и сравнительно низкой стоимости.

Момент электромагнитный, развиваемый асинхронным двигателем, определяется выражением

$$M = \frac{3 \cdot U_{Ф}^2 \cdot R_2' \ / s}{\omega_0 \cdot \left[\left(R_1 + \frac{R_2'}{s} \right)^2 + \left(X_1 + X_2^{'} \right)^2 \right]} = \frac{3 \cdot U_{Ф}^2 \cdot R_2' \ / s}{\omega_0 \cdot \left[\left(R_1 + \frac{R_2'}{s} \right)^2 + X_k^2 \right]}, \qquad (3.1)$$

где $U_Ф$ – действующее значение фазного напряжения сети, В;

R_1, R'_2 – активные сопротивления обмоток соответственно фазы статора и фазы ротора, приведенные к цепи статора, Ом;

X_1, X'_2 – индуктивные сопротивления обмоток соответственно фазы статора и фазы ротора, приведенные к цепи статора, Ом;

X_k – индуктивное фазное сопротивление короткого замыкания, Ом;

s – скольжение двигателя;

ω_0 – угловая скорость поля двигателя, рад/с.

Индуктивное фазное сопротивление короткого замыкания

$$X_k = X_1 + X'_2 \ . \tag{3.2}$$

Скольжение двигателя

$$s = \frac{\omega_0 - \omega}{\omega_0}. \tag{3.3}$$

Угловая скорость поля

$$\omega_0 = \frac{2 \cdot \pi \cdot f}{p}, \tag{3.4}$$

где f – частота напряжения питающей сети, Гц;

p – число пар полюсов двигателя.

Номинальную угловую скорость двигателя можно определить, зная номинальную частоту вращения n_{HOM}, об/мин, по выражению

$$\omega_{HOM} = \frac{\pi \cdot n_{HOM}}{30}. \tag{3.5}$$

Номинальное скольжение двигателя

$$s_{HOM} = \frac{\omega_0 - \omega_{HOM}}{\omega_0}. \tag{3.6}$$

Статическую механическую характеристику АД $\omega = f(M)$ (здесь M – момент электромагнитный) можно построить, используя формулу (3.1) и уравнение связи между угловой скоростью вала двигателя и скольжением.

$$\omega = \omega_0 \cdot (1 - s). \tag{3.7}$$

С учетом того, что момент электромагнитный по формуле (3.1) имеет экстремум (критическое значение), то возможна и другая форма записи зависимости $M = f(s)$, называемая формулой Клосса:

$$M = \frac{2 \cdot M_k \cdot (1 + a \cdot s_k)}{\frac{s}{s_k} + \frac{s_k}{s} + 2 \cdot a \cdot s_k}, \tag{3.8}$$

где M_k – максимальное (критическое) значение момента, Н·м;

s_k – критическое скольжение, соответствующее M_k;

a – параметр, $a = R_1/R'_2$.

Момент критический и скольжение критическое определяются следующими выражениями:

$$M_k = \pm \frac{3 \cdot U_Ф^2}{2 \cdot \omega_0 \cdot \left(R_1 \pm \sqrt{R_1^2 + X_k^2}\right)}; \tag{3.9}$$

$$s_k = \pm \frac{R'_2}{\sqrt{R_1^2 + X_k^2}}. \tag{3.10}$$

Для асинхронных двигателей большой мощности ($P_{2ном}$ > 100 кВт) сопротивление R_1 невелико. Тогда можно считать, что $R_1 << X_k$ и $a \cdot s_k << 1$. Уравнение механической характеристики в этом случае будет определяться выражениями

$$M = \frac{2 \cdot M_k}{\frac{s}{s_k} + \frac{s_k}{s}}; \quad s_k = \pm \frac{R'_2}{X_k}; \quad M_k = \pm \frac{3 \cdot U_Ф^2}{2 \cdot \omega_0 \cdot X_k}; \quad \omega = \omega_0 \cdot (1 - s).$$

Обычно в каталогах на асинхронные двигатели не приведены параметры схемы замещения (активные и индуктивные сопротивления фаз обмоток), поэтому вышеуказанные формулы имеют ограниченное применение для расчета статических характеристик АД. В каталогах на асинхронные двигатели, помимо номинальных данных ($P_{2ном}$, $n_{ном}$, $\cos\varphi_{ном}$, $\eta_{ном}$ и др.), приводится также кратность максимального момента в двигательном режиме по отношению к номинальному моменту $\mu_k = M_k/M_{ном}$. Кратность максимального момента иногда обозначают $\lambda = \mu_k$. Умножив номинальный момент двигателя на кратность максимального момента, нетрудно найти значение критического момента.

Значение s_k в каталогах не приводится, но оно может быть найдено по известным параметрам асинхронного двигателя.

При этом для машин малой мощности, у которых $R_1 \approx R'_2$, можно принять $a \approx 1$. Тогда

$$s_k = s_{ном} \cdot \frac{\lambda + \sqrt{\lambda^2 - 1 + 2 \cdot s_{но} \cdot (\lambda - 1)}}{1 - 2 \cdot s_{ном} \cdot (\lambda - 1)}. \tag{3.11}$$

В то же время для машин большой мощности, у которых $R_1 \approx 0$, можно принять $a \approx 0$. Тогда

$$s_k = s_{НОМ} \cdot \left(\lambda + \sqrt{\lambda^2 - 1}\right). \quad (3.12)$$

Таким образом, можно рассчитать статическую механическую характеристику асинхронного двигателя $\omega = f(M)$ по каталожным данным.

Чтобы определить скорость, с которой будет вращаться двигатель при заданном моменте нагрузки на валу, в формулу (3.8) необходимо подставить вместо M заданное значение момента статического M_c и решить его относительно скольжения. Следует иметь в виду, что решение даст два значения скольжения. Необходимо для дальнейших расчетов выбрать нужное значение s (оно должно быть больше нуля, но меньше s_k). Затем для выбранного значения s определить угловую скорость двигателя.

3.3 Содержание отчета

Отчет по работе № 3 оформляется индивидуально на листах формата А4 на бумажном или электронном носителе и имеет следующий состав:

- титульный лист;
- текст индивидуального задания;
- исходные данные задания;
- расчеты, выполненные по пп. 3.1.1–3.1.6.

Контрольные вопросы

1 Изложите методику расчета естественных электромеханической и механической характеристик асинхронного электродвигателя.

2 Изложите методику расчета искусственных электромеханической и механической характеристик асинхронного электродвигателя при изменении напряжения, подводимого к статору электродвигателя.

3 Изложите методику расчета искусственных электромеханической и механической характеристик асинхронного электродвигателя при изменении напряжения и частоты, подводимых к статору электродвигателя.

4 Изложите методику расчета ПИД-регулятора.

5 Поясните особенности цифровой реализации ПИД-регулятора.

4 Практическая работа № 4. Язык программирования LD

Цель работы:

– изучение средств описания данных и операций языка программирования LD;

– получение практических навыков программирования на языке LD задач автоматизации мехатронных систем.

4.1 Задание

1 Изучить работу типовых блоков, реализующих функции таймеров, счетчиков, триггеров, детекторов фронтов и т. д., входящих в библиотеку Standart.lib системы CoDeSys.

2 Изучить примеры применения типовых функциональных блоков в управляющих программах на языке LD.

3 Проверить работу программ, приведенных в примерах, в режиме эмуляции.

4 Записать программы в память контроллера и проверить их выполнение.

5 Составить отчет по работе.

4.2 Теоретические сведения

Существует набор типовых функциональных блоков, реализующий функции, часто используемые в программировании ПЛК. Это счетчики, таймеры, триггеры, детекторы фронтов и т. д. В стандартной библиотеке подпрограмм Standart.lib, входящей в комплект CoDeSys, к таким блокам относятся:

– таймеры (TON, TOF, TP);

– счетчики (CTU, CTD, CTUD);

– триггеры (RS, SR);

– детекторы фронтов (R_TRIG, F_TRIG).

Эти функциональные блоки могут использоваться в программах на языке LD совместно с типовыми элементами этого языка.

Триггеры.

RS- и SR-триггеры отличаются лишь реакцией сигналов на оба входа: SET и RESET. Напоминаем, что для классического RS-триггера такое состояние входов является запрещенным, т. к. приводит к неоднозначности выходного сигнала.

Для исследования этих триггеров достаточно набрать лишь два фрагмента, в которых участвуют указанные функциональные блоки. На рисунке 4.1 изображены фрагменты схем с RS- и SR-триггерами.

При замыкании контакта *X* подается сигнал на вход SET каждого из триггеров (см. рисунок 4.1), что вызывает срабатывание реле L и через замыкающий контакт L этого реле приходит сигнал на обмотку реле M, включающего, например, электрическую машину. Последующие размыкания

или замыкания не меняют состояние выходов этих триггеров, и катушки L и M остаются включенными. Для отключения реле M необходимо кратковременно нажать кнопку Res.

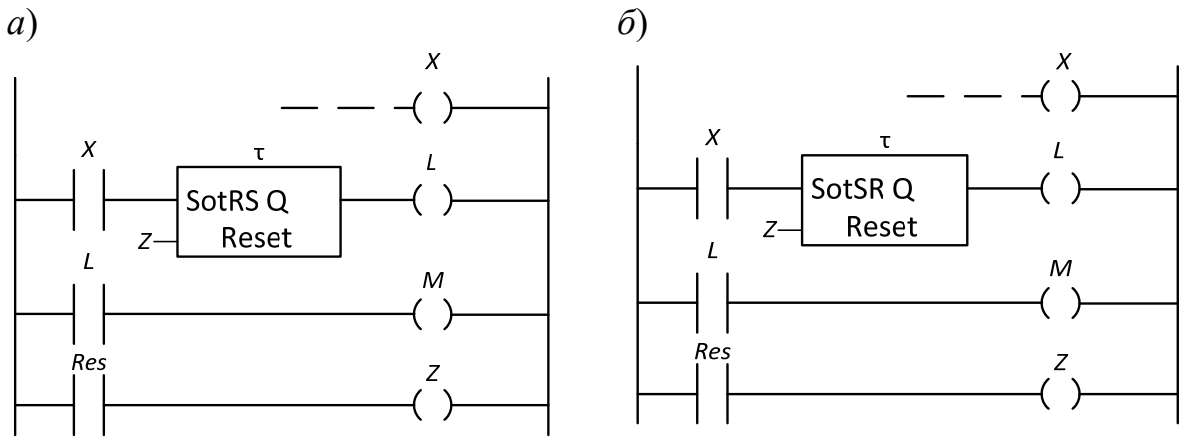

Рисунок 4.1 – Фрагменты схем с RS- и SR-триггерами

Если одновременно замкнутся контакты X и Res, то в RS-триггере преобладающим будет сигнал на отключение L и, соответственно, M, а в SR-триггере – на включение этих реле.

Необходимо напомнить, что входу RESET этих триггеров необходимо присвоить идентификатор (имя) того реле, которое будет обеспечивать сброс. В нашем примере в третьей цепи каждого фрагмента (см. рисунок 4.1) стоит реле с именем «Z». Поэтому на входах RESET поставлен тот же символ – «Z».

Детекторы импульсов.

Функциональные блоки R_TRIG и F_TRIG являются детекторами импульсов. Первый из них генерирует одиночный импульс по переднему фронту, а другой – по заднему спаду входного сигнала. Временные диаграммы входных и выходных сигналов показаны на рисунке 4.2.

Таймеры.

Три типа таймеров находят широкое применение:

1) TP-таймер или генератор одиночного импульса заданной длительности;

2) TOF-таймер с задержкой выключения;

3) TON-таймер с задержкой включения.

У этих таймеров есть вход IN для логических сигналов, вход PT для установки требуемых временных параметров и логический выход Q.

Работу этих таймеров поясняют временные диаграммы.

Из этих диаграмм следует, что TP-таймер запускается мгновенно передним фронтов входного сигнала и в течение времени *T* действия выходного сигнала не реагирует на новые импульсы, поступающие на вход IN.

TOF-таймер также срабатывает по фронту входа IN. Выход *Q* сбрасывается после спада входного сигнала с задержкой времени *T* от установленной по входу PT. Пауза между входными сигналами должна быть не меньше времени задержки.

TON-таймер срабатывает по переднему фронту входа IN, но сигнал на выходе *Q* появится с задержкой TB, установленной по входу PT.

Таймер не реагирует на импульсы продолжительностью менее значения TB.

При включении любого таймера в цепь многоступенчатой схемы (рисунок 4.2) программа запрашивает имя этого функционального блока (вопросительные знаки над таймером) и временную уставку по входу PT (вопросительные знаки у входа PT).

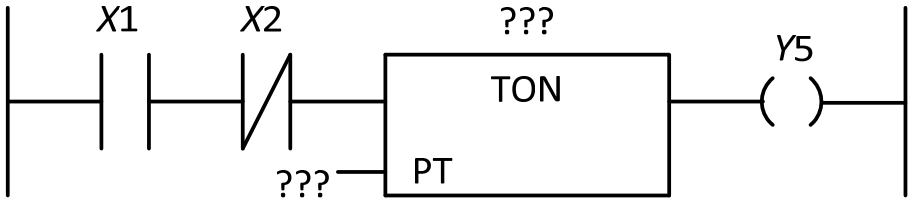

Рисунок 4.2 – Фрагмент схемы со вставленным таймером

Щелкнув левой клавишей мыши по верхним знакам ???, присваиваем имя. Например, N1. Щелкнув по знакам ??? у хода PT, нажать Shift, и не отпуская нажать T, затем #, отпустить Shift, нажать требуемое значение задержки (например, 15), единицу времени (например, s, т. е. секунды) и Enter. Этот фрагмент будет выглядеть, как показано на рисунке 4.3.

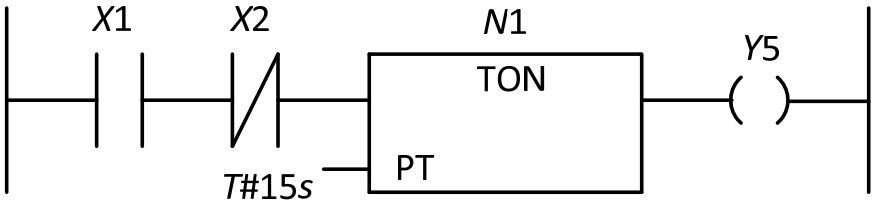

Рисунок 4.3 – Фрагмент схемы после записи уставок

Временные уставки задают в миллисекундах (ms), секундах (s), минутах (m) или часах (h). Уставка может быть дробной. К примеру, T#1,5m.

Счетчики.

CTU – инкрементный счетчик, CTD – декрементный; CTUD – инкрементный/декрементный.

Простейшая схема с CTU-счетчиком показана на рисунке 4.4.

После переноса счетчика в цепь многоступенчатой схемы появятся знаки ??? над блоком, на входе RESET и PV.

Знаки ??? над блоком заменяем именем. Знаки ??? перед PV запрашивают значение уставки, т. е. требуемое количество импульсов, вызывающее срабатывание счетчика, при котором выход Q перейдет в TRUE (при условии, что на входе RESET был сигнал FALSE).

Вход RESET знаками ??? запрашивает имя логического элемента, от которого должен поступить сигнал TRUE, останавливающий счет, обнуляющий

выход CV и устанавливающий на выходе Q FALSE.

В схеме (см. рисунок 4.4) для счетчика присвоено имя W, для входа RESET имя реле h и принята уставка PV=100.

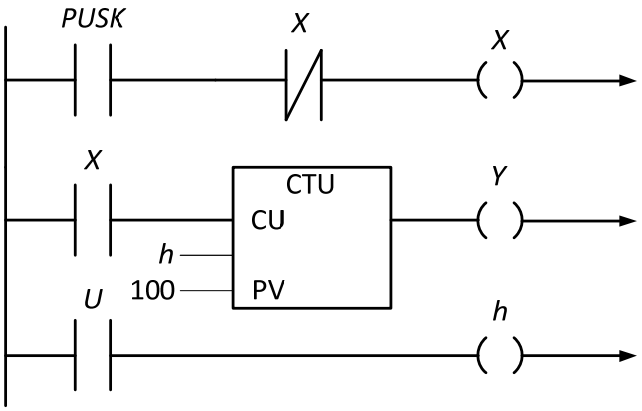

Рисунок 4.4 – Схема с CTU-счетчиком

Само реле h находится в третьей цепи и управляется кнопкой U.

Первая цепь содержит кнопку PUSK. По каждому фронту сигнала, поступающему на вход CU, значение выхода CV возрастает на единицу и как только их сумма достигнет значения PV, счет останавливается.

На других языках программирования ПЛК с выхода CV можно снимать информацию о количестве поступивших импульсов для последующей обработки с помощью операторов и функций.

CTD-счетчик отличается CTU тем, что каждый входной импульс уменьшает значение счетчика на единицу. Когда счетчик достигнет нуля, выход Q устанавливается в TRUE.

CTUD-инкрементный/декрементный счетчик имеет как накопительный вход CU (как в CTU), так и вычитающий CD (как в CTD).

Информация о работе данных блоков содержится в Руководстве пользователя по программированию ПЛК в CoDeSys 2.3 и в Справочной системе комплекса программирования CoDeSys.

Далее приведены примеры программ на языке LD, составленные с использованием этих функциональных блоков.

Пример 1 – Демонстрация работы реверсивного счетчика и детекторов фронтов.

Создать на языке LD программу, которая увеличивает на единицу значение целой переменной при наличии положительного фронта на дискретном входе 0 и уменьшает на единицу значение этой переменной при наличии отрицательного фронта на входе 1. Общий вид программы на языке LD представлен на рисунке 4.5. Для регистрации фронтов использованы детекторы фронтов R_TRIG и F_TRIG, для работы с целой переменной используется реверсивный счетчик CTUD. На вход CLK детектора фронтов подается дискретный сигнал: информация с дискретного входа, значение логической переменной, или

логического выражения. Выход Q детектора фронта устанавливается в единицу в том случае, если входное значение блока изменилось по сравнению со значением в предыдущем цикле, единичное значение сохраняется в течение одного цикла. R_TRIG выдает единицу, когда ноль на входе сменяется единицей, F_TRIG выдает единицу, когда единица на входе сменяется нулем. Переменные A и B связаны с дискретными входами.

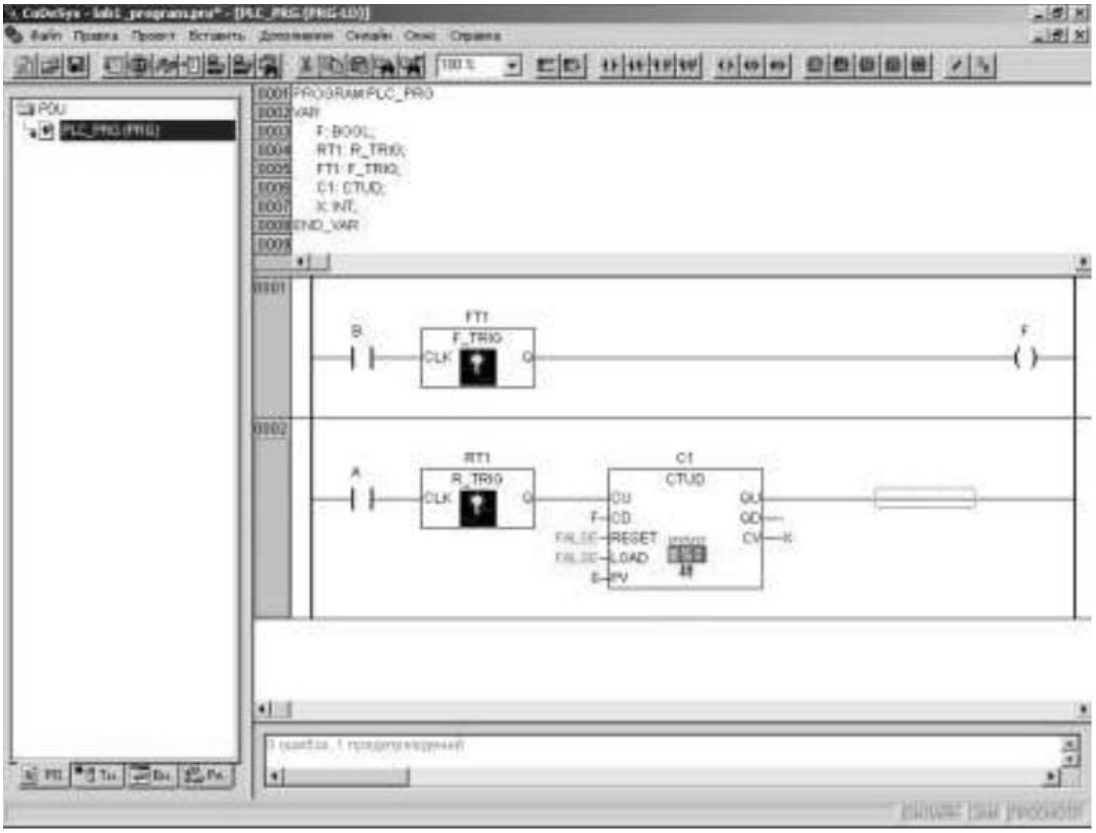

Рисунок 4.5 – Программа демонстрация работы детекторов фронтов и реверсивного счетчика

С первого дискретного входа значение сигнала подается на вход блока R_TRIG, объявленного как переменная RT1, со второго – на F_TRIG, объявленный, как переменная FT1. Выход FT1 связан с переменной F, которая далее подана на вход CD (уменьшение на единицу) счетчика. Выход RT1 подан напрямую на вход CU (увеличение на единицу) счетчика.

Переменная X, объявленная, как целое число, связана со счетным выходом счетчика CV. Выходы сброса счетчика на ноль (RESET) и загрузки в него начального значения (LOAD) в данном примере не используются, и на них подается логический ноль – логическая константа «ложь» – FALSE.

Поскольку счетчик CTUD используется не полностью, при компиляции данный пример сгенерирует одно предупреждение, но, несмотря на это, пример работает нормально. Тестирование примера просто: если нажимается кнопка, подключенная к первому входу, переменная X увеличивается на единицу, если нажимается и отпускается кнопка, подключенная ко второму входу, переменная X уменьшается на единицу. Следует обратить внимание, что детекторы

фронтов, счетчики и таймеры не являются базовыми (т. е. непредставимыми простыми операторами) программными единицами.

Все функциональные блоки можно реализовать программно с помощью базовых операций. В доказательство этого составим программу для того же самого примера без применения счетчиков и детекторов фронтов. Программа показана на рисунке 4.6.

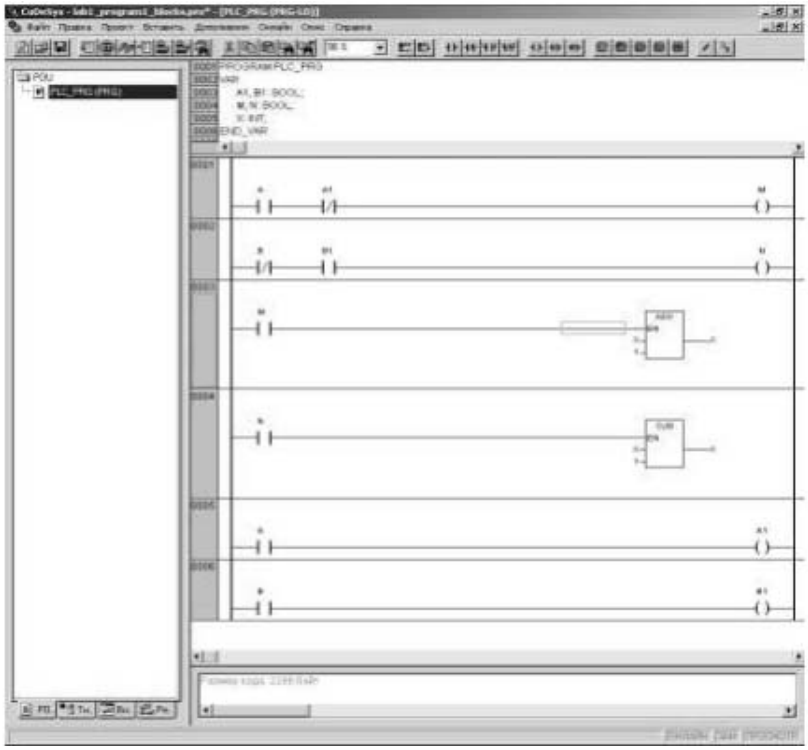

Рисунок 4.6 – Программа демонстрация работы детекторов фронтов и реверсивного счетчика

Стоит сказать об арифметических операциях. Они реализованы в виде стандартных функций ADD (сложение) и SUB (вычитание). При значении логической единицы на входе EN блок работает, не работает при логическом нуле на входе EN. Во встроенной справке приведен перечень всех операций, осуществляемых с помощью стандартных функций с входом EN.

Детектор фронта в этом примере реализован следующим образом: объявлены две дополнительные переменные, по одной на каждый детектируемый сигнал. В самом конце программы, после использования текущих значений сигналов, они сохраняются в объявленные переменные, и значения переменных используются в следующем цикле программы как значения, сохраненные в прошлом цикле, и так происходит каждый цикл.

Первые две строки программы представляют собой именно детектирование сигнала, единовременную проверку его значения в прошлом и настоящем шагах.

Пример 2 – Управление освещением в комнате.

Условие: есть комната, в двери стоят два датчика регистрации пересечения линии (снаружи и внутри комнаты), они подсоединены к ПЛК. Также к ПЛК подсоединен включатель комнатного освещения, есть возможность использовать еще одну кнопку. Требуется составить программу, которая управляет автоматическим включением и выключением света в комнате. Программа, являющаяся решением задачи, показана на рисунке 4.7.

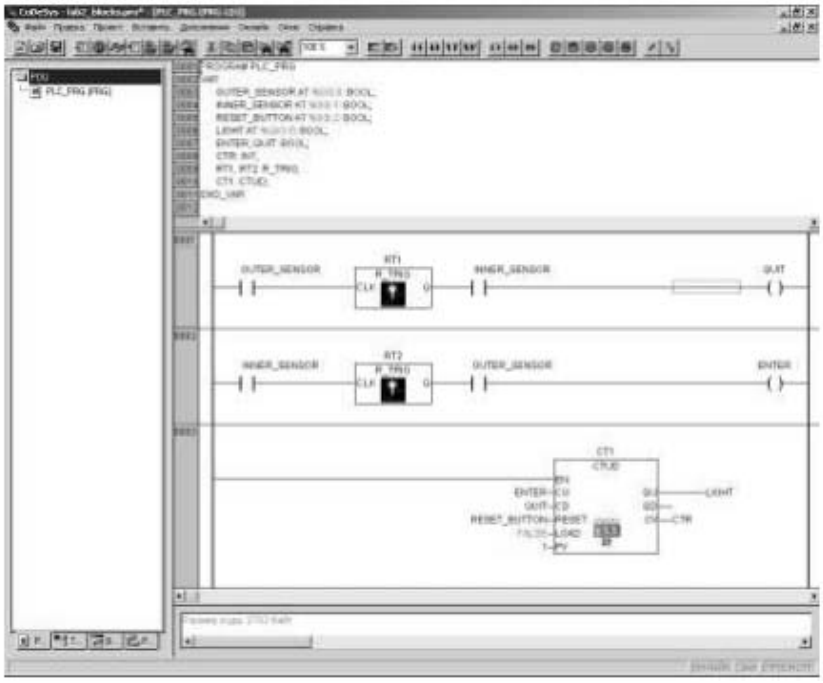

Рисунок 4.7 – Решение задачи об автоматическом включении света, сделанное с помощью типовых функциональных блоков

Если человек входит в комнату, то он пересекает сначала наружный датчик, потом внутренний, и в момент пересечения внутреннего датчика внешний датчик уже регистрирует присутствие человека в дверях. Процесс выхода из комнаты относительно датчиков происходит также, только датчики следует поменять местами. Таким образом, по переднему фронту одного датчика в сочетании с уже сработавшим другим получим короткий импульс, обозначающий вход, или выход одного человека. Далее требуется реализовать счет людей, это можно сделать с помощью реверсивного счетчика.

Также, если значение счетчика больше, или равно единице, следует включить свет, если нет, то выключить. Предположим, что возможна ситуация, когда человек, находясь в комнате, хочет выключить свет, для этого необходимо иметь кнопку принудительного гашения света, которую следует соединить со сбросом счетчика.

Приведем назначение переменных. OUTER_SENSOR и INNER_SENSOR – переменные, связанные с внутренним и наружным датчиком пересечения линии. Устанавливаются, если линия пересечена посторонним объектом и сбрасываются, если пересечения нет. RESET_BUTTON кнопка гашения света. QUIT

и ENTER – внутренние переменные, устанавливающиеся в единицу в моменты, соответственно, выхода из комнаты и входа в нее. LIGHT – переменная, связанная с реле включения света, CTR – переменная счетчика вошедших в комнату.

На рисунке 4.8 изображено решение той же задачи, но без применения типовых функциональных блоков.

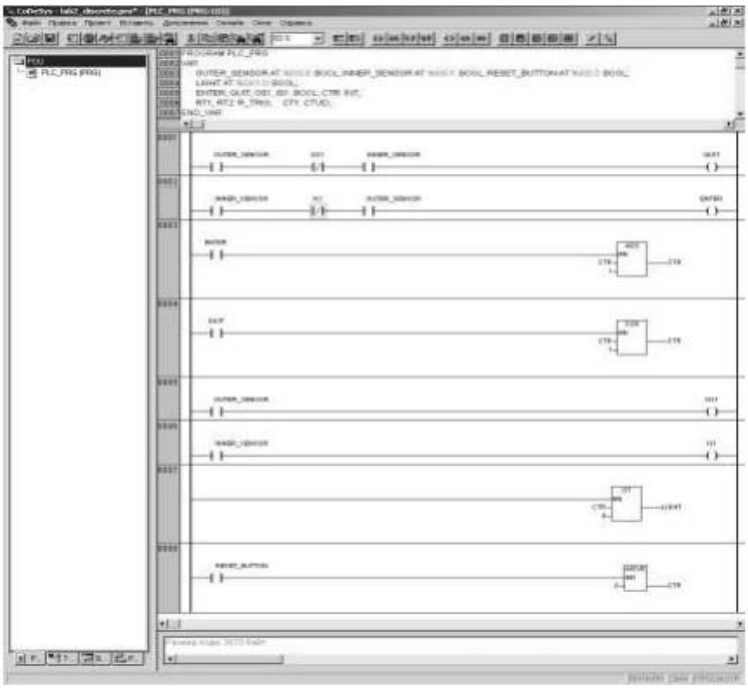

Рисунок 4.8 – Решение задачи об автоматическом включении света без типовых функциональных блоков

Переменные IS1 и OS1 хранят значения переменных INNER_SENSOR и OUTER_SENSOR за предыдущий цикл.

Функция GT – сравнение двух чисел на входе, и, если «верхнее» больше, чем «нижнее», функция возвращает логическую единицу. MOVE – пересылка значения. Слева указывается его источник, а справа – приемник, значение может быть любого типа, переменные источника и приемника должны быть одного и того же, или совместимых типов.

4.3 Содержание отчета

Отчеты по работе № 4 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе со следующим содержанием:

- титульный лист;
- текст индивидуального задания;
- исходные данные задания работы;
- разработанная программа на языке LD;
- описание результатов работы программы.

Контрольные вопросы

1 Поясните назначение языка LD.
2 Для каких задач лучше подходит язык LD?
3 Какие основные элементы могут использоваться при составлении программы на языке LD?
4 Как реализуются таймеры на языке LD?
5 Как реализуются триггеры на языке LD?

5 Практическая работа № 5. Язык программирования FBD

Цель работы:
– изучение средств описания данных и операций языка программирования FBD;
– получение практических навыков программирования на языке FBD задач автоматизации мехатронных систем.

5.1 Задание

1 Изучить работу типовых блоков, реализующих функции регулирования, входящих в библиотеку Util.lib, системы CoDeSys.
2 Изучить примеры применения типовых функциональных блоков языка FBD в управляющих программах.
3 Проверить работу программ, приведенных в примерах, в режиме эмуляции.
4 Записать программы в память контроллера и проверить их выполнение.
5 Составить отчет по работе.

5.2 Теоретические сведения

ПИД-регулирование является наиболее точным и эффективным методом поддержания контролируемой величины на заданном уровне. На рисунке 5.1 приведена функциональная схема ПИД-регулятора. Основное назначение регулятора – формирование управляющего сигнала Y, задающего выходную мощность исполнительного механизма (ИМ) и направленного на уменьшение рассогласования E, или отклонения текущего значения регулируемой величины T от величины уставки $T_{уст}$.

ПИД-регулятор состоит из трех основных частей: пропорциональной, интегральной и дифференциальной. Пропорциональная составляющая зависит от рассогласования E_i и отвечает за реакцию на мгновенную ошибку регулирования.

Интегральная составляющая содержит в себе накопленную ошибку регулирования и позволяет добиться максимальной скорости достижения

заданного значения.

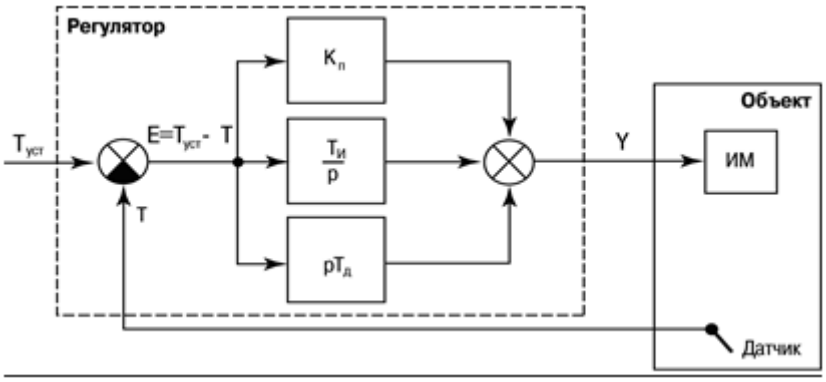

Рисунок 5.1 – Схема ПИД-регулятора

Дифференциальная составляющая зависит от скорости изменения рассогласования и позволяет улучшить качество переходного процесса.

В системе программирования CoDeSys в библиотеке UTIL.lib реализован стандартный блок ПИД-регулятора, который показан на рисунке 5.2.

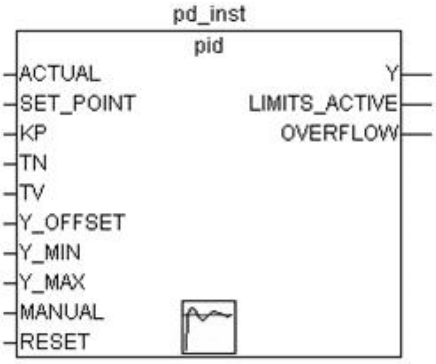

Рисунок 5.2 – ПИД-регулятор в CoDeSys

Функциональный блок реализует ПИД-закон регулирования:

$$Y = Y\_OFFSET + KP\left(e(t) + \frac{1}{TN}\int_0^{TN} e(t) + TV\frac{de(t)}{dt} \right), \qquad (5.1)$$

где Y_OFFSET – стационарное значение;

KP – коэффициент передачи;

TN – постоянная интегрирования, мс;

TV – постоянная дифференцирования, мс;

e(t) – сигнал ошибки (SET_POINT-ACTUAL).

Входы ACTUAL, SET_POINT, KP, Y_OFFSET, Y_MIN Y_MAX типа REAL.

Входы TN и TV типа DWORD, RESET и MANUAL типа BOOL.

Выходы Y – REAL, LIMITS_ACTVE и OVERFLOW типа BOOL.

Значение выхода Y ограничено Y_MIN и Y_MAX. При достижении Y границ ограничения выход LIMITS_ACTVE (BOOL) принимает значение TRUE. Если ограничение выхода не требуется, Y_MIN и Y_MAX должны быть равны нулю.

Неправильная настройка регулятора может вызвать неограниченный рост интегральной составляющей. Для обнаружения такой ситуации предназначен выход OVERFLOW. При переполнении он принимает значение TRUE, одновременно останавливается работа регулятора. Для его включения необходимо использовать рестарт.

Пример реализации ПИД- и двухпозиционного регулятора

Список переменных (рисунок 5.3) включает в себя параметры двух регуляторов и входы-выходы системы управления.

Программа управления на языке FBD, состоящая из двух функциональных блоков, представлена на рисунке 5.4.

```
0001 PROGRAM PLC_PRG
0002 VAR
0003 (*параметры ПИД-регулятора*)
0004     T:REAL:=20;(*ustavka*)
0005     ti:REAL:=1;(*integr.postyannaya*)
0006     td:REAL:=0;(*dif.postyannaya*)
0007     Xp:REAL:=10;(*polosa proporcionalnosty*)
0008     C1:REAL:=0;(*nizhnaya ustavka komparatora*)
0009     C2:REAL:=20;(*verhnaya ustavka komparatora*)
0010
0011 (*параметры для ПИД-регулятора*)
0012     a12:REAL:=0.1;(*zona nechustvitelnosty*)
0013     a13:REAL:=100;(*ogranicheniy vyh. moshnosty, %*)
0014     a14:BOOL;(*type isp. mechanizma: 0 - nagrevatel; 1 - ohladitel;*)
0015
0016 (*параметры для двухпозиционного регулятора*)
0017     a21:WORD:=0;(*type logiki: 0- vykl; 1 - prym.gysterezis;2 - obr.gysterezis; 3- P-logika; 4 - U-logika*)
0018     a29:BOOL;(*sostoyaniye VU2 pri neispravnosty: 0 - otkl;1 - vkl 100%*)
0019
0020 (*рабочие параметры*)
0021     dat:REAL;(*signal datchika temperature*)
0022     vu2:BOOL;(*signal vu2*)
0023     pvu1:REAL;(*vyh. moshnost signal vu1*)
0024     reg1: reg_2pos_cfc;
0025 END_VAR
0026 VAR
0027     pidreg: PID;
0028 END_VAR
0029 VAR
0030     reg2: reg_2pos;
0031 END_VAR
```

Рисунок 5.3 – Область объявления переменных проекта

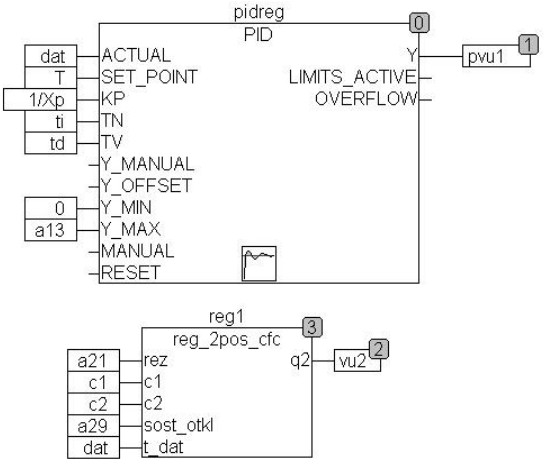

Рисунок 5.4 – Программа регулятора в CoDeSys

В алгоритме ПИД-регулятора реализовано ограничение по мощности регулирования (0 %...100 %).

5.3 Содержание отчета

Отчеты по работе № 5 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе со следующим содержанием:

– титульный лист;

– текст индивидуального задания;

– исходные данные задания работы;

– программа на языке FBD, реализующая управление с использованием ПИД-регулятора.

Контрольные вопросы

1 Поясните назначение языка FBD.

2 Для каких задач лучше подходит язык FBD?

3 Какие основные элементы могут использоваться при составлении программы на языке FBD?

4 Как реализуются таймеры на языке FBD?

5 Как реализуются триггеры на языке FBD?

6 Практическая работа № 6. Язык программирования ST

Цель работы:

– изучение средств описания данных и операций языка программирования ST;

– получение практических навыков программирования на языке ST задач автоматизации мехатронных систем.

6.1 Задание

1 Изучить работу типовых блоков, реализующих функции регулирования, входящих в библиотеку Util.lib, системы CoDeSys.

2 Изучить примеры применения типовых функциональных конструкций языка ST в управляющих программах.

3 Проверить работу программ, приведенных в примерах, в режиме эмуляции.

4 Записать программы в память контроллера и проверить их выполнение.

5 Составить отчет по работе.

6.2 Теоретические сведения

Стандартные операторы и реализация последовательного управления.

Основой ST-программы служат выражения. Результат вычисления выражения присваивается переменной при помощи оператора «:=», как и в Паскале. Каждое выражение обязательно заканчивается точкой с запятой «;». Выражение состоит из переменных констант и функций, разделенных операторами:

```
dblVar := 1 + intVar / abs(intVar);
```

Стандартные операторы в выражениях ST имеют символьное представление, например математические действия: +, *, /, операции сравнения и т. д.

Помимо операторов, элементы выражения можно отделять пробелами и табуляциями для лучшего восприятия. В текст могут быть введены комментарии. Везде, где допустимы пассивные разделители, можно вставлять и комментарии:

```
dblVar := 1 + (*получить знак*) intVar / abs(intVar); (*проверка на 0 была выше*)
```

Несколько выражений можно записать подряд в одну строку. Но хорошим стилем считается запись одного выражения в строке. Длинные выражения можно перенести на следующую строку. Перенос строки равноценен пассивному разделителю.

Выражение может включать другое выражение, заключенное в скобки. Выражение, заключенное в скобки, вычисляется в первую очередь.

Тип выражения определяется типом результата вычислений:

```
blnAlarm := byInp1 > bylnp2 AND bylnp1 + bylnp2 <> 0 OR blnAlarm2;
```

Вычисление выражения происходит в соответствии с правилами приоритета операций. Первыми выполняются операции с наивысшим приоритетом.

Операторы условного управления.

Оператор IF.

Оператор выбора (IF) позволяет выполнить различные группы выражений в зависимости от условий, выраженных логическими выражениями. Полный синтаксис оператора IF (если) выглядит так:

```
IF <логическое_выражение_IF> THEN
<выражения_IF>;
{
ELSIF <логическое_выражение_ELSEIF_1> THEN
<выражения_ELSEIF_1>;
…
ELSIF <логическое_выражение_ELSEIF_n> THEN
<выражения_ELSEIF_n>;
```

ELSE
<выражения_ELSE>;
}
END_IF;

Если <логическое_выражение_IF> ИСТИНА, то выполняются выражения первой группы – <выражения_IF>. Прочие выражения пропускаются, альтернативные условия не проверяются.

Часть конструкции в фигурных скобках является необязательной и может отсутствовать.

Если <логическое_выражение_IF> ЛОЖЬ, то одно за другим проверяются условия ELSIF. Первое истинное условие приведет к выполнению соответствующей группы выражений. Прочие условия ELSIF анализироваться не будут. Групп ELSIF может быть несколько или не быть совсем.

Если все логические выражения дали ложный результат, то выполняются выражения группы ELSE, если она есть. Если группы ELSE нет, то не выполняется ничего.

Например:

IF temp < 17
THEN heating_on: = TRUE;
ELSE heating_on: = FALSE;
END_IF

В этом примере нагревание (heating) включается, когда температура опустится ниже 17 º, иначе оно останется выключенным.

Оператор выбора CASE.

С помощью инструкции CASE можно нескольким различным значениям целочисленной переменной сопоставить различные инструкции.

Полный синтаксис оператора CASE выглядит следующим образом:

CASE <переменная> OF
<значение1>: <выражения_1>
<значение2>: <выражения_2>
<значение3, значение4, значение5>: <выражения_3>
<значение6 .. значение10>: <выражения_4>
…
<значение_n>: <выражения_n>
ELSE <выражения_ELSE>
END_CASE;

Операторы циклического управления

Цикл FOR

Цикл FOR обеспечивает заданное количество повторений группы

выражений. Синтаксис:

FOR <целочисленный_счетчик> := <начальное_значение> TO <конечное_значение> {BY <шаг>} DO
<выражения — тело цикла>
END_FOR

Перед выполнением цикла счетчик получает начальное значение. Далее тело цикла повторяется, пока значение счетчика не превысит конечного значения. Счетчик увеличивается в каждом цикле. Начальное и конечное значения и шаг могут быть как константами, так и выражениями.

Счетчик изменяется после выполнения тела цикла. Поэтому, если задать конечное значение меньшее начального, то при положительном приращении цикл не будет выполнен ни разу. При одинаковых начальном и конечном значениях тело цикла будет выполнено один раз.

Часть конструкции BY в скобках необязательна, она определяет шаг приращения счетчика. По умолчанию счетчик увеличивается на единицу в каждой итерации. В качестве счетчика можно использовать переменную любого целого типа.

Пример:

```
intVar := 0;
FOR cw := 1 TO 10 DO
intVar := intVar + 1;
END_FOR
```

Данный цикл будет выполнен 10 раз и соответственно intVar будет иметь значение 10.

Цикл FOR исключительно удобен для итераций с заранее известным числом повторов.

Цикл WHILE.

Циклы WHILE обеспечивает повторение группы выражений, пока верно условное логическое выражение.

Синтаксис WHILE:

WHILE <условное_логическое_выражение> DO <выражения — тело цикла>
END_WHILE

Условие в цикле WHILE проверяется до начала цикла. Раздел <выражения> выполняется циклически до тех пор, пока <условное_логическое_выражение> дает ИСТИНА (TRUE). Если логическое выражение изначально имеет значение ЛОЖЬ (FALSE), тело цикла не будет выполнено ни разу. Если <условное_логическое_выражение> никогда не примет значение ЛОЖЬ

(FALSE), то раздел <выражения> будет выполняться бесконечно.

Пример:

```
ci := 64;
intVar := 0;
WHILE ci > 1 DO
intVar := intVar + 1;
ci := ci / 2;
END_WHILE
```

Цикл REPEAT.

Циклы REPEAT также, как и цикл WHILE, обеспечивает повторение группы выражений, пока верно условное логическое выражение. Цикл REPEAT отличается от цикла WHILE тем, что первая проверка условия выхода из цикла осуществляется, когда цикл уже выполнился 1 раз. Это означает, что независимо от условия выхода цикл выполняется хотя бы один раз.

Синтаксис REPEAT:

```
REPEAT
<выражения — тело цикла >
UNTIL <условное_логическое_выражение>
END_REPEAT
```

Условие в цикле REPEAT проверяется после каждого выполнения тела цикла. Раздел <выражения> выполняется циклически до тех пор, пока <условное_логическое_выражение> дает ИСТИНА (TRUE). Если логическое выражение изначально имеет значение ЛОЖЬ (FALSE), тело цикла будет выполнено один раз. Если <условное_логическое_выражение> никогда не примет значение ЛОЖЬ (FALSE), то раздел <выражения> будет выполняться бесконечно.

Оператор EXIT и RETURN.

Оператор EXIT, помещенный в теле циклов WHILE, REPEAT и FOR, приводит к немедленному окончанию цикла. Хороший стиль программирования призывает избегать такого приема, но иногда он весьма удобен.

Оператор RETURN осуществляет немедленный возврат из POU. Это единственный способ прервать вложенные итерации без введения дополнительных проверок условий. Оператор RETURN выполняется очень быстро, фактически это одна машинная команда процессора.

Иногда бывает удобно создать безусловный цикл, а условия выхода формировать в теле цикла с использованием EXIT. Например, могут потребоваться несколько равновероятных, но невзаимосвязанных условий выхода из цикла. Создать безусловный (бесконечный) цикл в ST проще всего так:

```
WHILE TRUE DO…
```

На рисунке 6.1 показан пример реализации ПИД-регулятора на языке ST.

```
0001 CASE rez OF
0002 (*регулятор выключен*)
0003 0:IF sost_otkl =0 THEN q2:=0;
0004 ELSE q2:=1;
0005 END_IF;
0006
0007  (*прямой гистерезис*)
0008 1: IF t_dat<c1 THEN q2:=1;
0009 END_IF;
0010 IF t_dat>c2 THEN q2:=0;
0011 END_IF;
0012
0013 (*обратный гистерезис*)
0014 2: IF t_dat>c2 THEN q2:=1;
0015 END_IF;
0016 IF t_dat<c1 THEN q2:=0;
0017 END_IF;
0018
0019 (*П-логика*)
0020 3: IF (t_dat>c1) AND (t_dat<c2) THEN q2:=1;
0021 ELSE q2:=0;
0022 END_IF;
0023
0024  (*U-логика*)
0025 4:IF (t_dat>c1) AND (t_dat<c2) THEN q2:=0;
0026 ELSE q2:=1;
0027 END_IF;
0028 END_CASE;
```

Рисунок 6.1 – Подпрограмма двухпозиционного регулятора на языке ST

6.3 Содержание отчета

Отчеты по работе № 6 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе со следующим содержанием:

- титульный лист;
- текст индивидуального задания;
- исходные данные задания работы;
- листинг разработанной программы на языке ST.

Контрольные вопросы

1 Поясните назначение языка ST.

2 Для каких задач лучше подходит язык ST?

3 Какие основные элементы могут использоваться при составлении программы на языке ST?

4 Как реализуются таймеры на языке ST?

5 Как реализуются триггеры на языке ST?

7 Практическая работа № 7. Применение SCADA в МС

Цель работы:

– изучение основных характеристик SCADA для программирования элеметов мехатронных систем;

– получение практических работы со SCADA мехатронных систем.

7.1 Задание

Создать визуализацию для проекта по заданию преподавателя. При создании визуализации предусмотреть индикацию состояния датчиков технологической установки, а также возможность управления установкой из визуализации.

7.2 Теоретические сведения

Визуализация представляет собой графическое изображение проектируемой системы, которое может служить пользовательским интерфейсом для контроля и управления работой системы.

Визуализация может исполняться в системе программирования, в отдельном приложении CoDeSys HMI, как веб-приложение на сервере или как целевая программа в ПЛК.

Редактор визуализации CoDeSys предоставляет набор готовых графических элементов, которые могут быть связаны соответствующим образом с переменными управляющей программы. Форма и цвет графических элементов могут изменяться при работе программы в зависимости от значений переменных.

Свойства отдельных элементов визуализации, а также визуализации в целом устанавливаются в соответствующих диалогах конфигурации и диалоге свойств объекта. Здесь определяется начальный вид элементов и выполняется привязка динамических свойств к значениям переменных проекта.

На рисунке 7.1 приведен пример визуализации.

Создание файла визуализации

Для создания визуализации нужно перейти на вкладку «Визуализации» организатора объектов, после чего правой кнопкой мыши вызвать контекстное меню в пустом поле и выбрать строку «Добавить объект», после чего появится окно визуализации (рисунок 7.2).

Элемент визуализации – это графический элемент, который используется при построении объекта визуализации. Возможные элементы представлены в виде иконок на панели инструментов CoDeSys (рисунок 7.3). Каждый элемент имеет собственную конфигурацию (набор свойств). Имеется возможность вставлять в визуализацию различные геометрические формы, а также точечные рисунки, метафайлы, кнопки и существующие визуализации.

Рисунок 7.1 – Пример визуализации программы контроллера

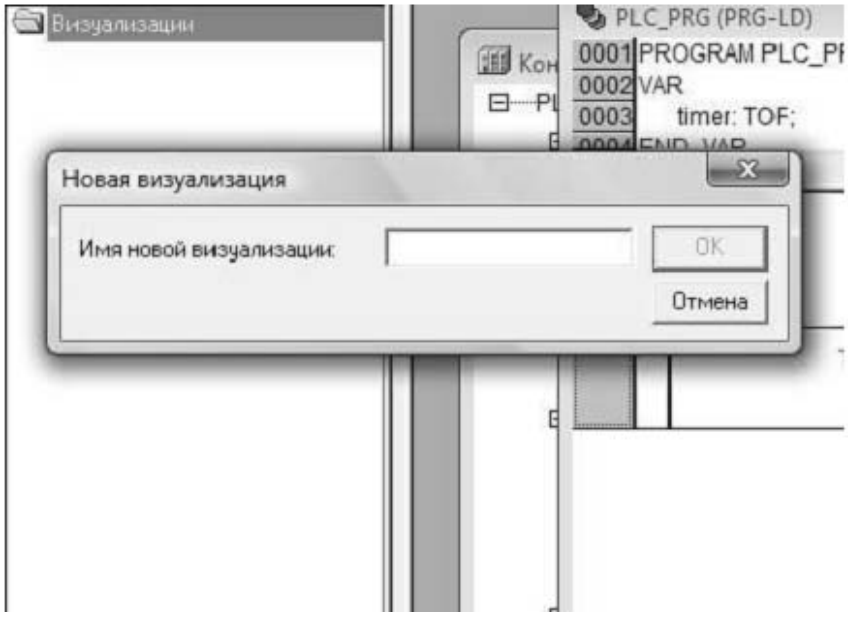

Рисунок 7.2 – Вызов окна визуализации

Рисунок 7.3 – Панель инструментов редактора визуализации

У каждого элемента визуализации есть свои свойства, вызвать свойства объекта визуализации можно двойным щелчком мыши по объекту либо активизировать правой кнопкой мыши объект в контекстном меню и выбрать строку «Конфигурировать». В открывшемся окне можно задавать необходимые свойства объекта визуализации (рисунок 7.4).

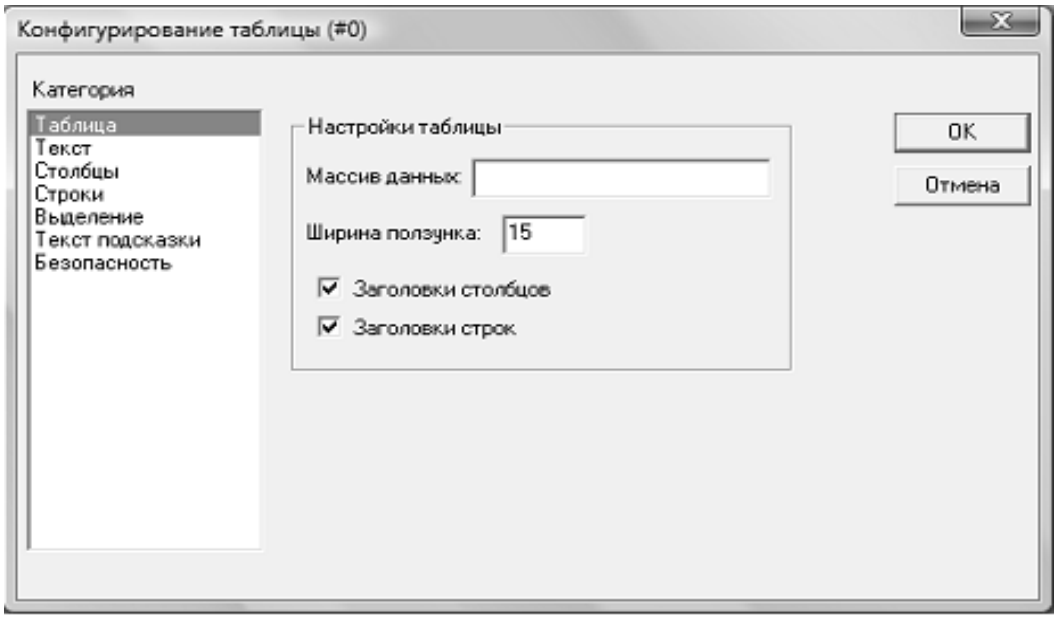

Рисунок 7.4 – Окно конфигурирования

Пример – Создание простейшей визуализации.

Создадим программу на языке LD в виде цепочки, состоящей из «Контакта» и «Катушки».

Перейдем на вкладку «Визуализация» и вызовем контекстное меню для добавления новой визуализации (рисунок 7.5).

В появившемся окне введем имя для визуализации (рисунок 7.6).

После ввода имени, появляется новое окно для создания визуализации.

Вставляем с помощью панели инструментов элементы «Эллипс» и «Кнопка», как на рисунке 7.7.

Теперь необходимо сконфигурировать эти элементы. Пусть необходимо сделать следующую визуализацию: эллипс должен менять цвет, когда срабатывает катушка реле, а при нажатии на кнопку должна изменять значение переменная типа BOOL.

Сначала добавляем переменную C типа BOOL в проект (рисунок 7.8).

Затем перейдем в окно редактора визуализации и щелкнем по эллипсу для вызова окна с настройками (рисунок 7.9).

Настроим категории «Цвета» и «Переменные». В категории «Цвета» выберем цвет эллипса в двух состояниях (рисунок 7.10).

Затем необходимо указать переменную, которая будет изменять цвет, для этого перейдем в категорию «Переменные» и установим курсор в поле «Изм. цвета:» (рисунок 7.11).

Рисунок 7.5 – Добавление объекта визуализации

Рисунок 7.6 – Ввод имени визуализации

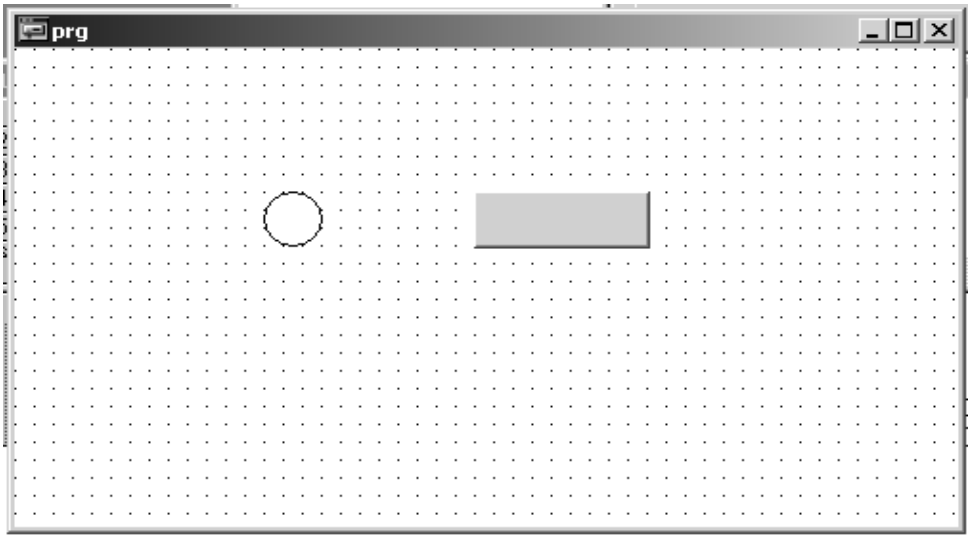

Рисунок 7.7 – Ввод элементов визуализации

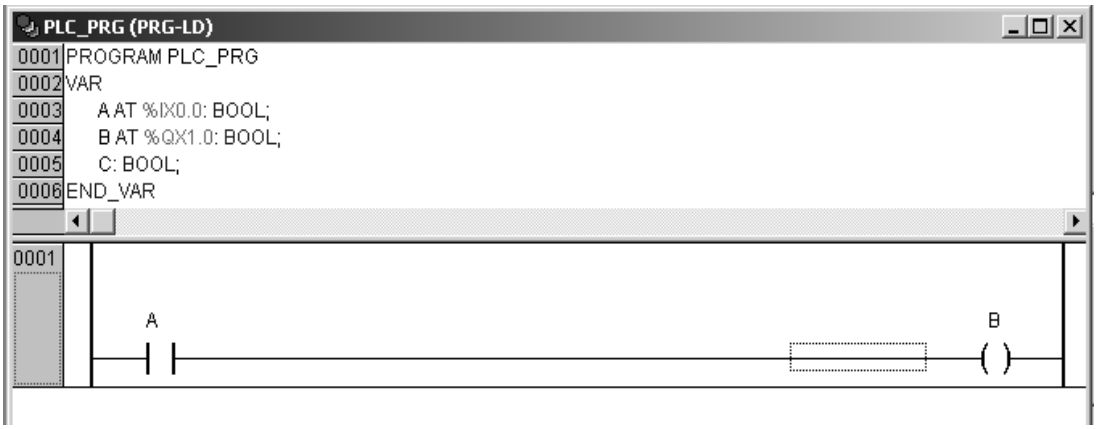

Рисунок 7.8 – Переменные проекта

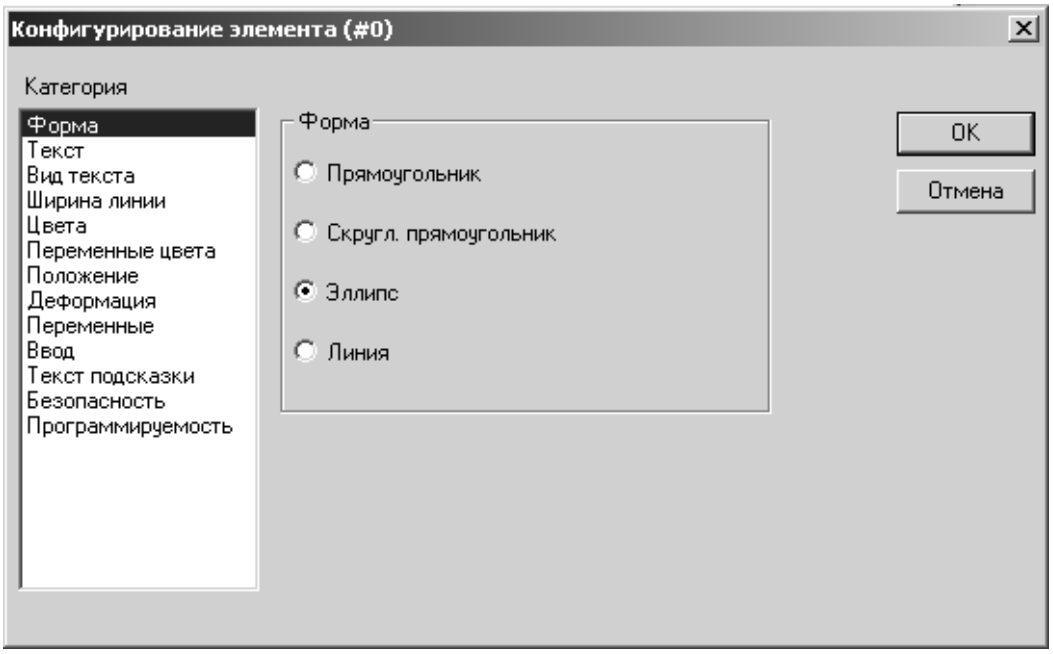

Рисунок 7.9 – Окно конфигурирования эллипса

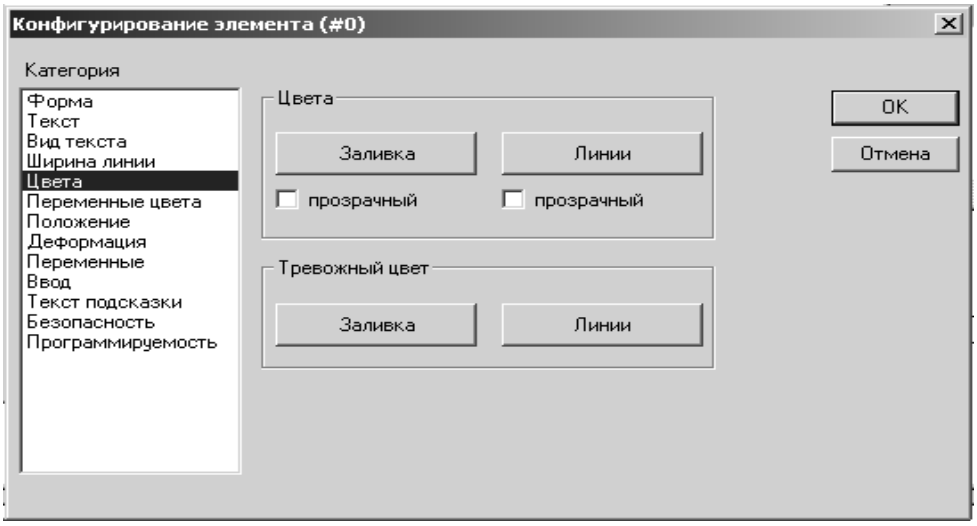

Рисунок 7.10 – Окно конфигурации цвета эллипса

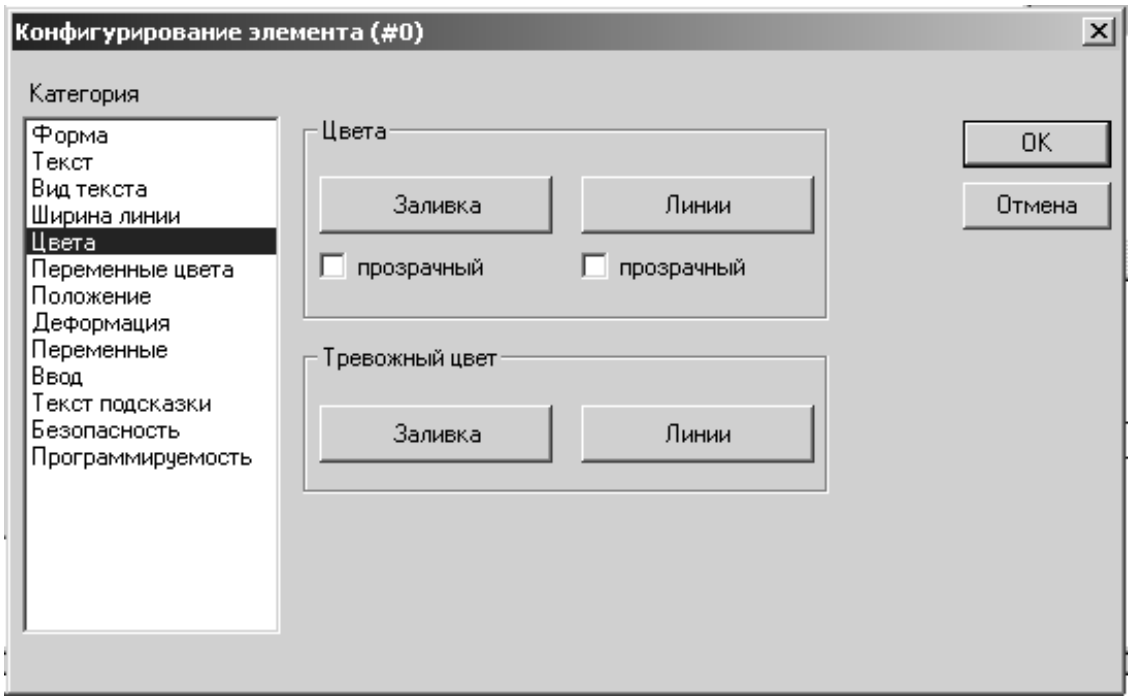

Рисунок 7.11 – Окно конфигурации переменных эллипса

Для связывания с переменными используем ассистент ввода (клавиша F2), в появившемся окне выберем нужную переменную – переменная B (рисунок 7.12).

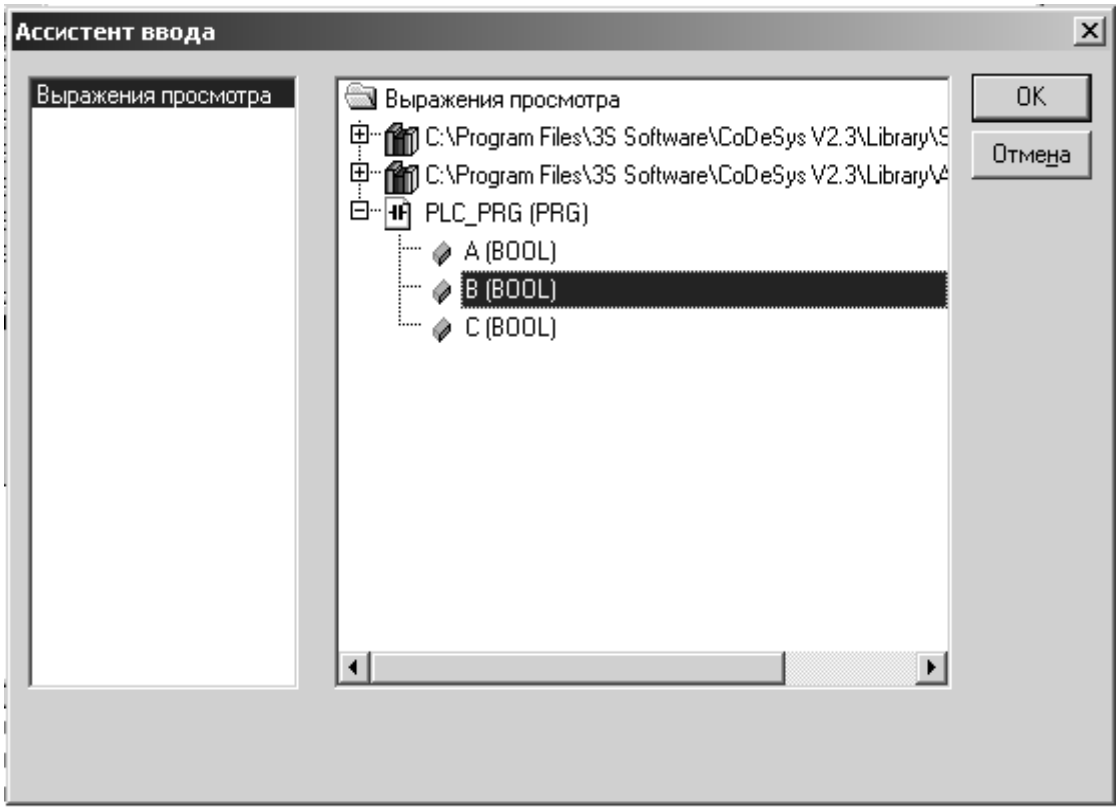

Рисунок 7.12 – Окно ассистента ввода

После нажатия кнопки ОК в окне конфигурации переменных появится переменная для изменения цвета (рисунок 7.13).

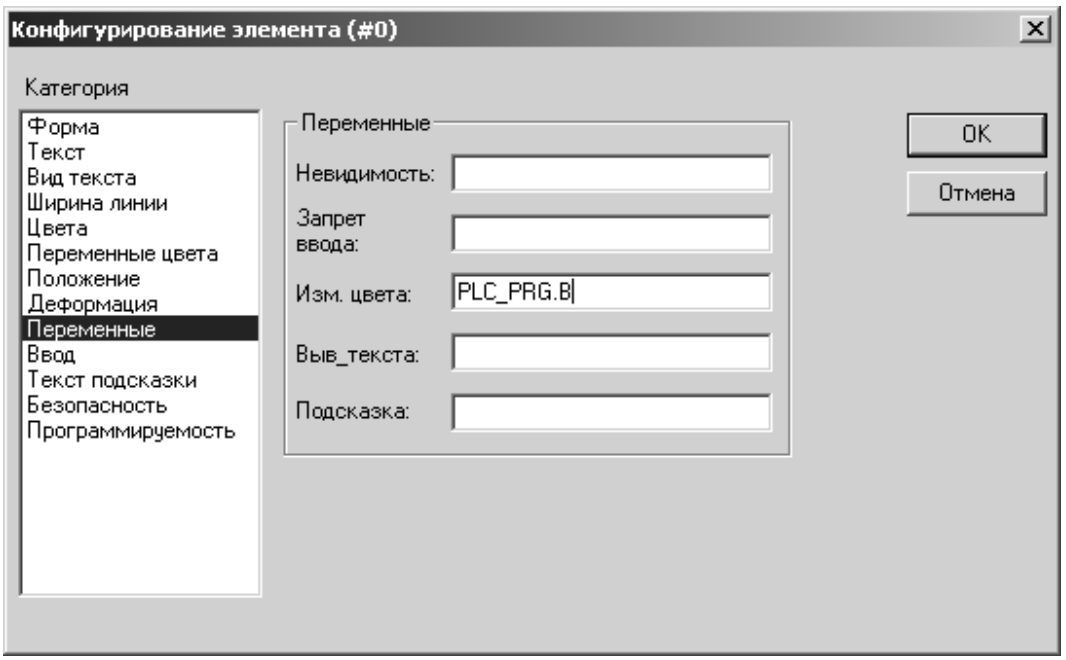

Рисунок 7.13 – Сконфигурированное окно переменных эллипса

Далее вызываем окно конфигурирования кнопки (рисунок 7.14) и выбираем категорию «Ввод».

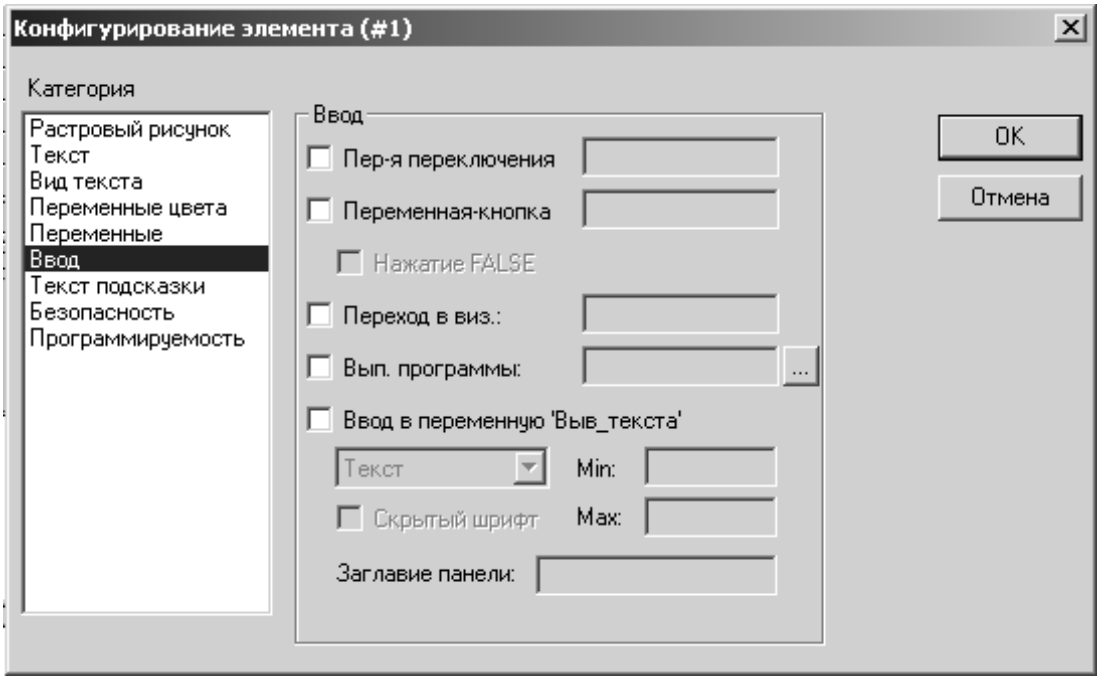

Рисунок 7.14 – Окно конфигурирования переменных кнопки

Поставим галочку «Переменная кнопка» (чтобы получить кнопку с фиксацией, необходимо выбирать «Переменная переключения») и, установив курсор в

появившееся поле ввода, вызовем ассистент ввода, чтобы привязать к кнопке переменную C (рисунок 7.15).

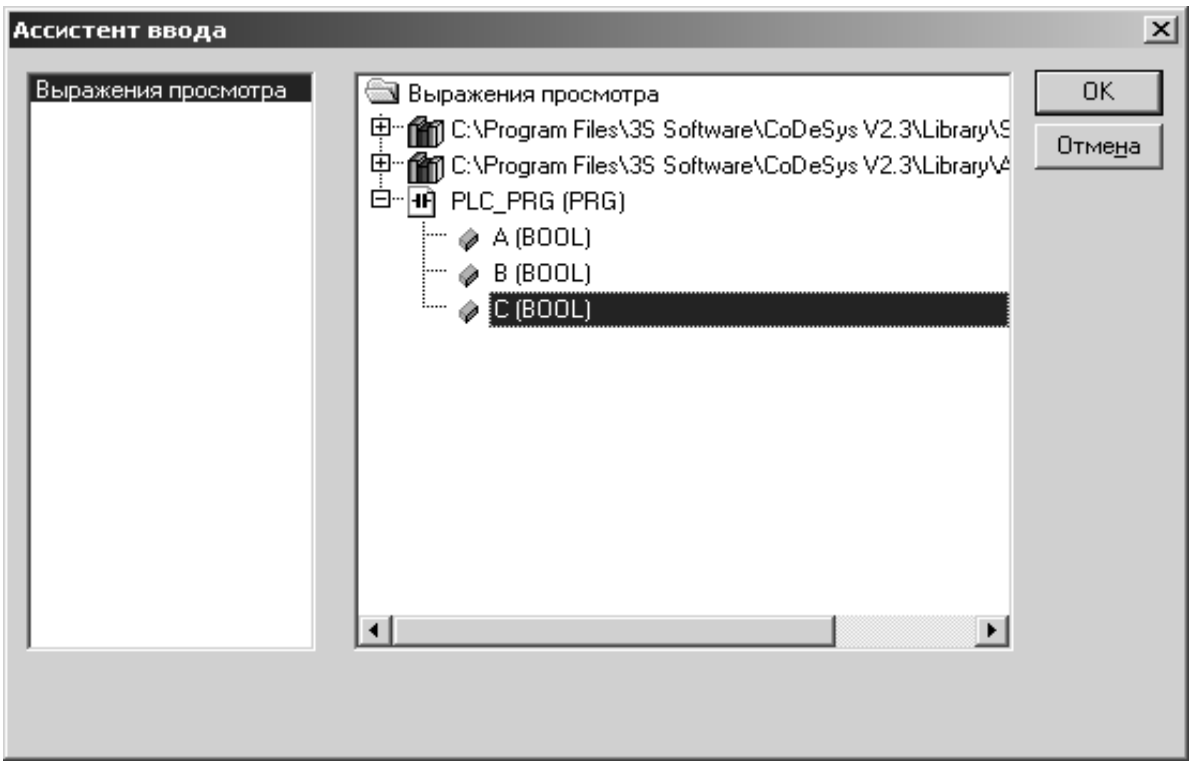

Рисунок 7.15 – Присваивание переменной

Сконфигурированное окно переменных показано на рисунке 7.16.

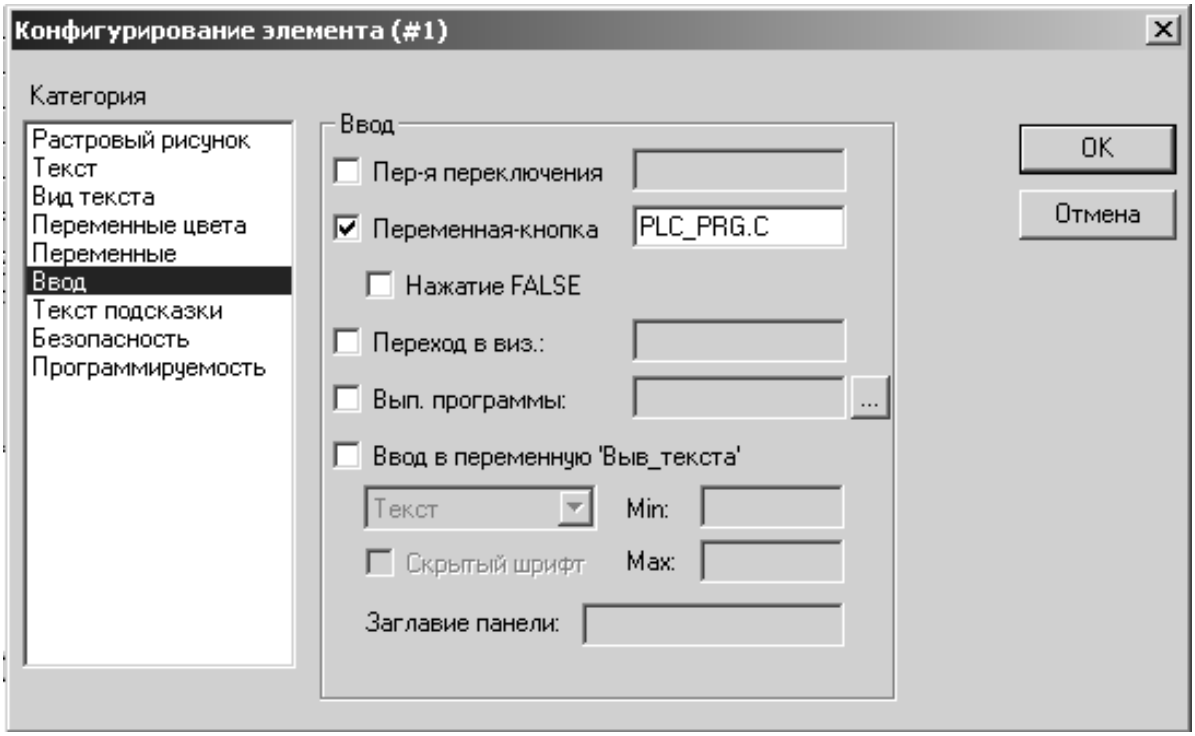

Рисунок 7.16 – Сконфигурированное окно переменных кнопки

Создание визуализации завершено. Можно загрузить программу в ПЛК и проверь работу визуализации.

7.3 Содержание отчета

Отчеты по работе № 7 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе со следующим содержанием:

– титульный лист установленного образца;
– цель работы;
– описание последовательности создания визуализации;
– список элементов визуализации в системы CoDeSys;
– ответы на контрольные вопросы;
– вывод по лабораторной работе.

Контрольные вопросы

1 Какие формы визуализации реализованы в системе CoDeSys?

2 Что такое объект визуализации и элемент визуализации в системе CoDeSys?

3 Какие элементы визуализации можно использовать при создании визуализаций в системе CoDeSys?

4 Каким образом осуществляются выбор и вставка элемента визуализации в системе CoDeSys?

5 Какие действия можно производить над элементом визуализации в системе CoDeSys?

6 Как одновременно переместить несколько элементом визуализации в системе CoDeSys?

7 Какие свойства элемента визуализации можно изменять при конфигурировании его в системе CoDeSys?

8 Как добавить текст к элементу визуализации в системе CoDeSys?

9 Какие свойства объекта визуализации можно изменять при конфигурировании его в системе CoDeSys?

10 Как установить фоновый рисунок при создании визуализации в системе CoDeSys?

8 Практическая работа № 8. ПО для анализа и синтеза МС

Цель работы:

– изучение основных видов программного обеспечения, используемого для анализа и синтеза мехатронных систем;

– получение навыков практической работы системами компьютерной математики для анализа и синтеза системы стабилизации скорости МС.

8.1 Задание

Осуществить синтез и анализ системы управления мехатронной системой в среде Simulink. Произвести настройку ПИ-регулятора в системе управления с использованием модуля Nonlinear Control Design.

8.2 Теоретические сведения

Пакет прикладных программ для построения нелинейных систем управления Nonlinear Control Design (NCD) Blockset реализует метод динамической оптимизации. Этот инструмент автоматически настраивает параметры моделируемых систем, основываясь на определенных пользователем ограничениях на их временные характеристики.

Пакет реализует следующие возможности:

– легкую настройку переменных;

– указание неопределенных параметров систем;

– интерактивную оптимизацию;

– моделирование методом Монте-Карло;

– поддержка проектирования как одномерных, так и многомерных систем управления.

Пакет расширения NCD является частью пакета Simulink и наследует все его приемы работы. Это, в частности, относится к вызову библиотек пакета NCD, их применению для построения моделей нелинейных систем и запуску процесса моделирования. Данный пакет является специализированной оптимизирующей программой для решения задачи оптимизации при наличии ограничений в форме неравенств и использующей в качестве алгоритма оптимизации последовательное квадратичное программирование.

Пакет NCD Blockset содержит следующие блоки:

– блок CRMS (Continuous RMS);

– блок DRMS (Discrete RMS);

– блок NCD Outport.

Блок CRMS реализует математическую зависимость

$$y(t) = \sqrt{\frac{1}{t}\int_0^t u^2(\tau)d\tau}\,, \tag{8.1}$$

где $y(t)$ – выходной сигнала блока;

t – время расчета, с;

$u(t)$ – входной сигнал.

При $t \to \infty$ $y(t)$ является среднеквадратическим (стандартным) отклонением.

Блок DRMS реализует такую же зависимость, что и блок CRMS, но для сигналов, определенных в дискретные моменты времени.

Рассматриваемые блоки могут применяться в системах моделирования, где качество функционирования целесообразно оценивать интегральным квадратичным критерием или стандартным отклонением ошибки.

Блок NCD Output является основным в рассматриваемом наборе блоков. Он имеет свое рабочее окно и меню и позволяет в интерактивном режиме выполнять следующие операции:

– задавать требуемые ограничения во временной области на любой сигнал оптимизируемой системы;

– указывать параметры, подлежащие оптимизации;

– указывать неопределенные параметры;

– проводить параметрическую оптимизацию системы с учетом заданных ограничений.

Типовой сеанс работы в среде Simulink с использованием возможностей и блоков NCD Blockset состоит из ряда стадий.

1 В среде Simulink создается модель исследуемой динамической системы (в общем случае нелинейной), для которой применяется метод интегрирования с переменным шагом.

2 Входы блоков NCD Outport соединяются с теми сигналами системы, на которые накладываются ограничения. Этими сигналами могут быть, например, выходы системы, их среднеквадратические отклонения и т. д.

3 Выполняется настройка блока NCD. Двойным щелчком на пиктограмме NCD Output данные блоки «раскрываются».

4 При помощи мыши нужным образом изменяются конфигурации и размеры областей ограничений для нужных сигналов системы. С помощью меню блока NCD Output задается интервал дискретизации (один или два процента от длительности процесса моделирования) и указываются имена (идентификаторы) параметров системы, подлежащих оптимизации.

5 Задаются неопределенные параметры системы, указываются их минимальные значения.

6 При необходимости сформированные ограничения сохраняются в виде файла с помощью команды меню Save (позднее они могут быть загружены с помощью команды Load).

7 Процесс оптимизации системы инициализируется нажатием кнопки Start.

Рассмотрим пример настройки ПИ регулятора при помощи блока NCD. Модель настраиваемой системы показана на рисунке 8.1.

В данном случае накладывается ограничение на регулируемую координату, поэтому соединим выход объекта регулирования со входом NCD Outport.

Вызовем меню настройки NCD (необходимо кликнуть по NCD outport). После чего появится окно настройки (рисунок 8.2).

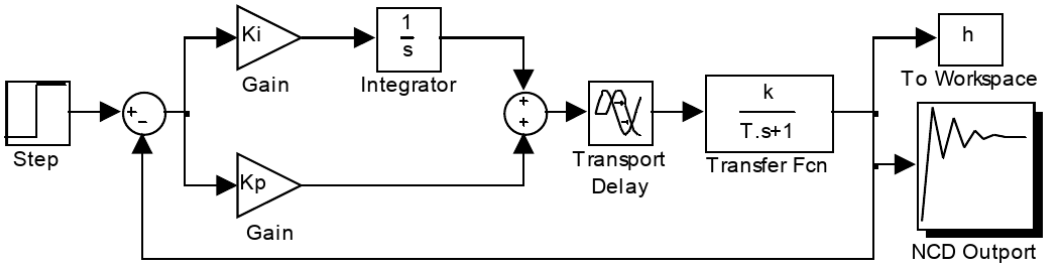

Рисунок 8.1 – Модель для настройки ПИ-регулятора

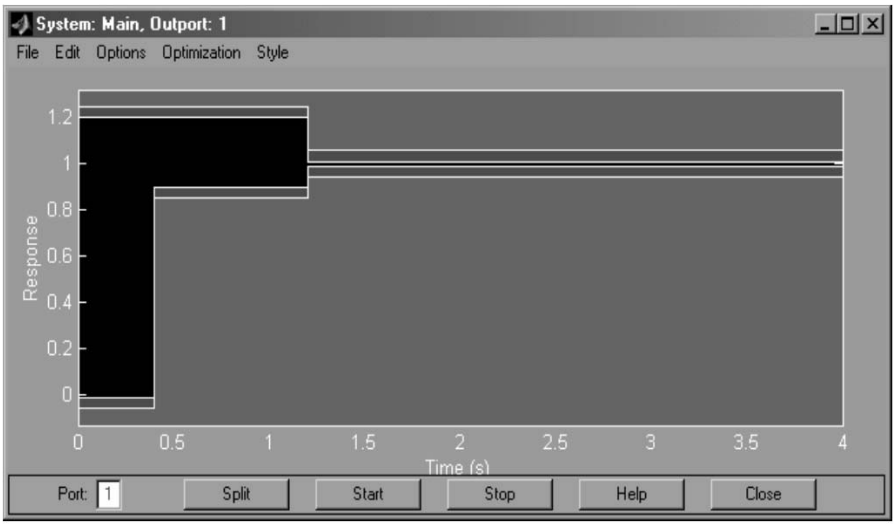

Рисунок 8.2 – Окно настройки блока NCD Outport

Первым этапом в настройке является настройка параметров желаемого переходного процесса (рисунок 8.3) (Options\Step response). Данное действие можно выполнить вручную, при помощи манипулятора (мышки).

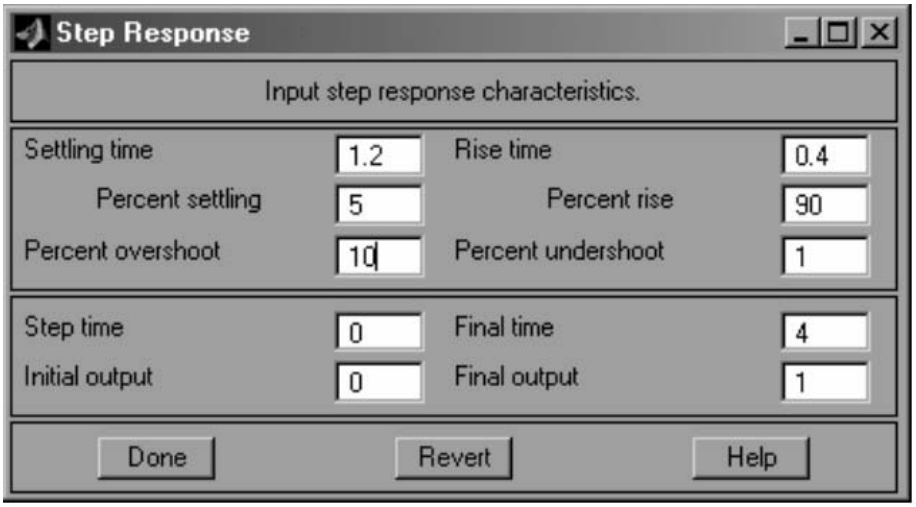

Рисунок 8.3 – Настройка параметров переходного процесса

Основные параметры формы Step Response:

– время завершения переходного процесса (Setting time);

– задание величины трубки в процентах, соответствующей установившемуся процессу (Percent setting);

– перерегулирование в процентах (Percent overshoot);

– время первого согласования (Rise time);

– недорегулирование в процентах (Percent undershoot);

– время начала действия входного сигнала (Step time);

– начальное значение (Initial output);

– время завершения моделирования (Final time);

– конечное значение входного сигнала (Final output).

Далее необходимо задать параметры, которые подлежат оптимизации, т. е. Kp, Ki. Для этого необходимо перейти в меню Optimization\Parameters (рисунок 8.4). В соответствии с рисунком 8.4 были заданы:

– настраиваемые коэффициенты (Tunable Variables) – Kp, Ki;

– нижний диапазон изменения настраиваемых коэффициентов (Lower bounds);

– верхний диапазон изменения настраиваемых коэффициентов (Upper bounds);

– период дискретизации (Discretization interval);

– переменные, определяющие точность (Variable Tolerance, Constraint Tolerance).

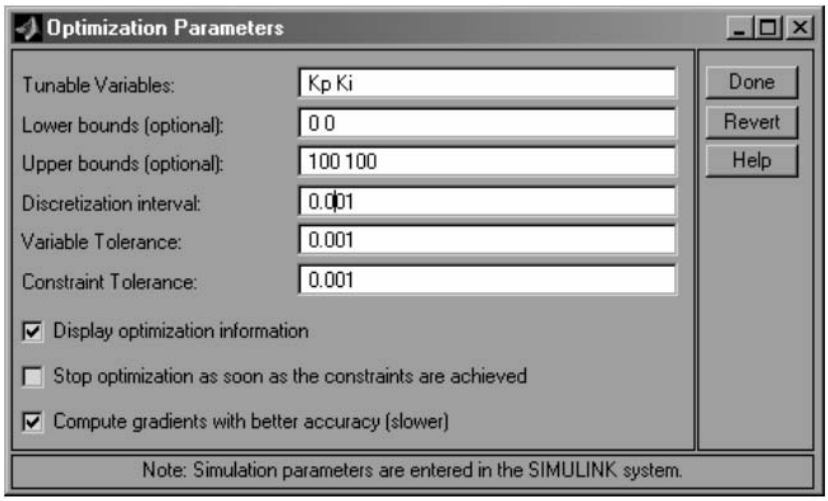

Рисунок 8.4 – Параметры оптимизации.

Также можно установить следующие параметры:

– отображать информацию об оптимизации (Display optimization information);

– остановить оптимизацию, как только достигнута переменная, определяющая точность расчета (Stop optimization as soon as the constraints are achieved);

– выполнять расчет градиента (Compute gradient with better accuracy).

Наименьшее значение коэффициентов регулятора должно гарантировать устойчивость замкнутой системы регулирования.

Рассматриваемый объект управления имеет неопределенности k и T. Рассмотрим, как выполняется настройка программы (рисунок 8.5).

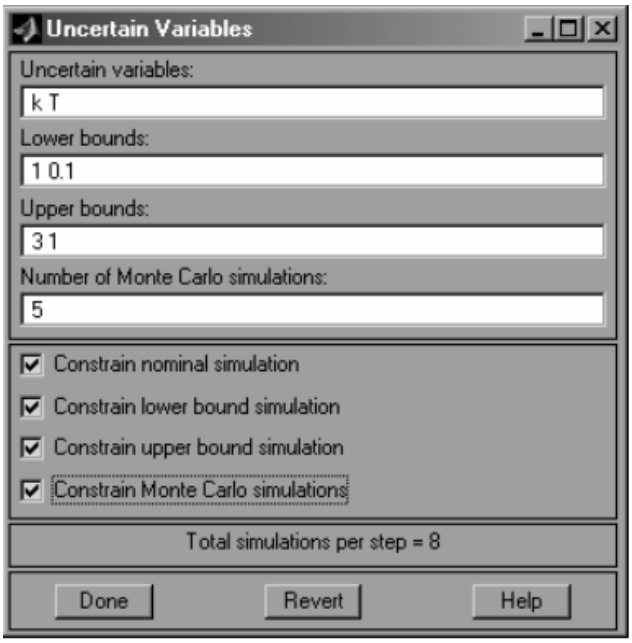

Рисунок 8.5 – Настройка параметров-неопределенностей.

В окне настройки параметров оптимизации необходимо задать следующие параметры:

– изменяющиеся переменные (Uncertain variable);

– нижний диапазон (Lower bounds). В форме заданы для k – 1, для T – 0,1;

– верхний диапазон (Upper bounds). В форме заданы для k – 3, для T – 1;

– число расчетов методом Монте-Карло (Number of Monte Carlo simulation);

– выполнять моделирование с переменными, заданными в рабочей области (Constrain nominal simulation);

– выполнять моделирование с переменными, соответствующими нижнему диапазону (Constrain lower bound simulation);

– выполнять моделирование с переменными, соответствующими верхнему диапазону (Constrain upper bound simulation).

– выполнять моделирование с переменными, полученными после расчета Монте-Карло (Constrain Monte Carlo simulation).

8.3 Содержание отчета

Отчеты по работе № 8 оформляются индивидуально на листах формата А4 в соответствии с требованиями ГОСТ 7.32 на бумажном или электронном носителе со следующим содержанием:

- титульный лист;
- текст индивидуального задания;

– исходные данные задания работы;
– результаты синтеза системы управления.

Контрольные вопросы

1 Для чего предназначен пакет прикладных программ Nonlinear Control Design (NCD) Blockset?
2 Каковы возможности Nonlinear Control Design Blockset?
3 Какие блоки включат Nonlinear Control Design?
4 Назначение блока CRMS.
5 Назначение блока DRMS.
6 Приведите отличия блоков CRMS и DRMS.
7 Назначение блока NCD Output.

Список литературы

1 **Шишов, О. В.** Программируемые контроллеры в системах промышленной автоматизации : учебник / О. В. Шишов. – Москва : ИНФРА-М, 2021. – 365 с.
2 **Гагарина, Л. Г.** Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; под ред. Л. Г. Гагариной. – Москва : ФОРУМ ; ИНФРА-М, 2019. – 400 с.
3 **Гагарина, Л. Г.** Введение в теорию алгоритмических языков и компиляторов : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева; под ред. Л. Г. Гагариной. – Москва : ФОРУМ, 2018. – 176 с.
4 **Мякишев, Д. В.** Принципы и методы создания надежного программного обеспечения АСУТП : учебное пособие / Д. В. Мякишев. – 2-е изд. – Москва; Вологда : Инфра-Инженерия, 2021. – 116 с.
5 **Ананьева, Т. Н.** Стандартизация, сертификация и управление качеством программного обеспечения : учебное пособие / Т. Н. Ананьева, Н. Г. Новикова, Г. Н. Исаев. – Москва : ИНФРА-М, 2021. – 232 с.
6 **Павлов, В. П.** Автоматизация моделирования мехатронных систем транспортно-технологических машин: учебное пособие / В. П. Павлов, А. Ю. Ахпашев. – Красноярск : СФУ, 2016. – 144 с.
7 **Матюшин, А. О.** Программирование микроконтроллеров: стратегия и тактика / А. О. Матюшин. – Москва : ДМК Пресс, 2017. – 356 с.
8 **Кангин, В. В.** Разработка SCADA-систем : учебное пособие / В. В. Кангин, М. В. Кангин, Д. Н. Ямолдинов. – Москва; Вологда: Инфра-Инженерия, 2019. – 564 с.
9 **Башлыков, А. А.** Основы конструирования интеллектуальных систем поддержки принятия решений в атомной энергетике : учебник / А. А. Башлыков, А. П. Еремеев. – Москва : ИНФРА-М, 2021. – 351 с.
10 **Осипова, Н. В.** Программное обеспечение систем управления : учебное пособие / Н. В. Осипова. – Москва : МИСиС, 2019. – 74 с.