Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра Информатики

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.И. Рубан
«27» июня 2016 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

231000.62 «Программная инженерия»

Инструментальная поддержка редактирования и валидации программ на языке программирования релейно-лестничной логики стандарта IEC 61131-3

Руководитель	_____	доцент, к. т. н.	А.С. Кузнецов
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		В.С. Гасенко
	подпись дата		инициалы фамилия
Нормконтроль	_____	доцент, к. т. н.	О.А. Антамошкин
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2016

Федеральное государственное автономное

образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ КОСМИЧЕСКИХ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра Информатики

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.И. Рубан
«20» мая 2016 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Гасенко Владиславу Сергеевичу

фамилия, имя, отчество

Группа КИ12-18Б Направление (специальность) 231000.62

номер код

Программная инженерия

наименование

Тема выпускной квалификационной работы Инструментальная поддержка редактирования и валидации программ на языке программирования релейно-лестничной логики стандарта IEC 61131-3

Утверждена приказом по университету № 6145/с от 10.05.2016

Руководитель ВКР В.С. Кузнецов, доцент, к.т.н

инициалы, фамилия, должность, ученое звание и место работы

Исходные данные для ВКР: Стандарт IEC 61131-3

Перечень разделов ВКР: Характеристики и анализ предметной области, Технологии разработки и ведения проекта, Техническое задание на разработку, Средства разработки программной системы, Принципы построения интерпретатора, Разработка программной системы, Руководство пользователя

Перечень графического материала: презентационные слайды PowerPoint

Руководитель ВКР _____ А.С. Кузнецов

подпись инициалы и фамилия

Задание принял к исполнению _____ Гасенко В.С.

подпись, инициалы и фамилия студента

« ___ » _____ 2016 г.

3

РЕФЕРАТ

Выпускная бакалаврская квалификационная работа по теме «Инструментальная поддержка редактирования и валидации программ на языке релейно-лестничной логики» содержит 39 страницы текстового документа, 21 рисунков, библиографических источников.

ЯЗЫК ПРОГРАММИРОВАНИЯ LADDER DIARGAM, IEC 61131-3, ИНТЕПРЕТАТОР, ТРАНСЛЯТОР XML, ПЛК.

Целью работы является проектирование и разработка программного комплекса для редактирования и валидации программ на языке Ladder Diagram (далее LD) для ПЛК работающих под управлением ОС Linux

В ходе данной работы был произведен обзор существующих средств трансляции языков стандарта IEC 61131-3. Было выяснено, что программные инструменты, которые являются интерпретаторами данных языков, практически отсутствуют.

Так же была описана структура программной системы для редактирования и валидации программ на языке релейно-лестничной логики (LD)

Был разработан интерпретатор программ на языке LD. Который включал в себя транслятор XML, фазы лексического и синтаксического анализа, а также процесс выполнения программы.

ОГЛАВЛЕНИЕ

1 ВВЕДЕНИЕ

Первые контроллеры появились на рубеже 60 – х и 70 – х годов в автомобильной промышленности, где использовались для автоматизации сборочных линий. В то время компьютеры стоили чрезвычайно дорого, поэтому контроллеры строились на жесткой логике (программировались аппаратно), что было гораздо дешевле. Однако перенастройка с одной технологической линии на другую требовала фактически изготовления нового контроллера. Поэтому появились контроллеры, алгоритм работы которых мог быть изменен несколько проще - с помощью схемы соединений реле. Такие контроллеры получили название программируемых логических контроллеров (ПЛК), и этот термин сохранился до настоящего времени

В настоящее время промышленность нельзя представить без систем автоматизации. Высокая сложность производственных процессов препятствует управление ими ручными, так же системы автоматики обходятся дешевле, чем обслуживающий персонал, кроме этого скорость и надежность выполнения действий выше.

Сейчас предприятие кроме полностью автоматизированными или роботизированными линиями включает в себя и отдельные полуавтономные участки – системы блокировки и аварийной защиты, системы подачи воды и воздуха, очистные сооружения, погрузочно-разгрузочные и складские терминалы и т.п. Функции автоматизированного управления для них выполняют программно-технические комплексы (ПТК). Они строятся с использованием аппаратно-программных средств, к которым относятся средства измерения и контроля и исполнительные механизмы, объединенные в промышленные сети и управляемые промышленными компьютерами с помощью специализированного ПО. При этом, в отличие от компьютерных сетей, центральным звеном ПТК

является не главный процессор, а программируемые логические контроллеры, объединенные в сеть

В качестве основного режима работы ПЛК выступает его длительное автономное использование, зачастую в неблагоприятных условиях окружающей среды, без серьёзного обслуживания и практически без вмешательства человека.

Целью работы является проектирование и разработка программного комплекса для редактирования и валидации программ на языке релейно-лестничной логики Ladder Diagram для ПЛК работающих под управлением ОС Linux.

Для достижения поставленной цели решались следующие задачи:

- Анализ систем трансляции языков стандарта IEC 61131-3;
- Обзор языка стандарта IEC 61131-3 Ladder Diagram;
- Проектирование интерпретатора языка Ladder Diagram;
- Программная реализация интепретатора языка Ladder Diagram.

Основными пользователями являются специалисты по автоматизации производственных процессов и программисты логических контроллеров.

Характеристики и анализ предметной области

В настоящее время для программирования ПЛК (программируемых логических контролеров) используются стандартизированные языки МЭК (IEC) стандарта IEC61131-3. Они в свою очередь делятся на два типа [1-2]:

- графические;

- текстовые.

Графических языков программирования стандарта IEC61131-3 три и еще один не сертифицирован:

- LD (Ladder Diagram) - язык релейных схем - самый распространённый язык для PLC;

- FBD (Function Block Diagram) - язык функциональных блоков - 2-й по распространённости язык для ПЛК;

- SFC (Sequential Function Chart) - язык диаграмм состояний - используется для программирования автоматов;

- CFC (Continuous Function Chart) - не сертифицирован IEC61131-3, дальнейшее развитие FBD;

Текстовых языков программирования стандарта IEC61131-3 два:

- IL (Instruction List) — Ассемблеро-подобный язык

- ST (Structured Text) — Паскале-подобный язык

В настоящее время программируемы логические контроллеры (далее ПЛК) активно используются во всех сферах человеческой деятельности. Для компиляции программ используемых в ПЛК нужен набор компиляторов проекта с кросс-компиляцией. Это введет в свою очередь к тому, что требуется перекомпилировать программу при каждом, даже незначительном изменении и иметь под рукой компилятор. компиляции программ, написанных на данных языках, нужен набор компиляторов проекта с кросс-компиляцией. Что в свою

очередь приводит к неудобствам, а именно, при внесении любых изменений, даже самых незначительных, нужно перекомпилировать всю программу для ПЛК, кроме этого, нужно всегда иметь под рукой компилятор. А использование компилятора ведет к тому, что для загрузки изменений, даже незначительных, его нужно подключать, что в свою очередь не удобно, если изменения частые и небольшие

Реализация интерпретатора на языке LD, позволит менять программу и вносить небольшие изменения без повторной компиляции целиком всей программы на самом ПЛК используя только графический редактор и сам интерпретатор, что поможет избавится в необходимости в наборе компиляторов проекта.

Устройство ПЛК

До появления твердотельных логических схем разработка систем логического управления основывались на электромеханических реле. По сей день реле не устарели в своем предназначении, но все же в некоторых своих прежних функциях они заменены контроллером.

В современной промышленности существует большое количество различных систем и процессов, требующих автоматизации, но теперь такие системы редко проектируются из реле. Современные производственные процессы нуждаются в устройстве, которое запрограммировано на выполнение различных логических функций. В конце 1960-х годов американская компания «Bedford Associates» разработала компьютерное устройство, названное MODICON (Modular Digital Controller). Позже название устройства стало названием подразделения компании, спроектировавшей, сделавшей и продавшей его [3].

Другие компании разработали собственные версии этого устройства, и, в конце концов, оно стало известно как ПЛК, или программируемый логический контроллер. Целью программируемого контроллера, способного имитировать

работу большого количества реле, была замена электромеханических реле на логические элементы.

ПЛК имеет набор входных клемм, с помощью которых можно контролировать состояние датчиков и выключателей. Также имеются выходные клеммы, которые сообщают «высокий» или «низкий» сигнал индикаторам питания, электромагнитным клапанам, контакторам, небольшим двигателям и другим самоконтролируемым устройствам.

ПЛК легки в программировании, так как их программный язык напоминает логику работы реле. Так обычный промышленный электрик или инженер-электрик, привыкший читать схемы релейной логики, будет чувствовать себя комфортно и при программировании ПЛК на выполнение тех же функций.

Подключение сигналов и стандартное программирование несколько отличаются у разных моделей ПЛК, но они достаточно схожи, что позволяет разместить здесь «общее» введение в программирование этого устройства.

На рисунке 1 представлен простой ПЛК, а точнее то, как он может выглядеть спереди. Две винтовые клеммы, обеспечивающие подключение для внутренних цепей ПЛК напряжением до 120 В переменного тока, помечены L1 и L2.

Шесть винтовых клемм, расположенных с левой стороны, обеспечивают подключение для входных устройств. Каждая клемма представляет свой входной канал (X). Винтовая клемма («общее» подключение) расположенная в левом нижнем углу обычно подключается к L2 (нейтральная) источника тока напряжением 120 В переменного тока.

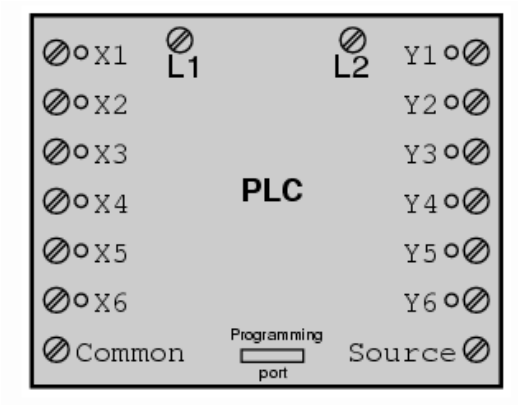

Рисунок 1 – Иллюстрация простого ПЛК

Внутри корпуса ПЛК, связывающего каждую входную клемму с общей клеммой, находится оптоизолятор устройства (светодиод), который обеспечивает электрически изолированный «высокий» сигнал для схемы компьютера (фототранзистор интерпретирует свет светодиода), когда 120-тивольтный переменный ток устанавливается между соответствующей входной клеммой и общей клеммой. Светодиод на передней панели ПЛК дает возможность понять, какой вход находится под напряжением – это показано на рисунке 2.

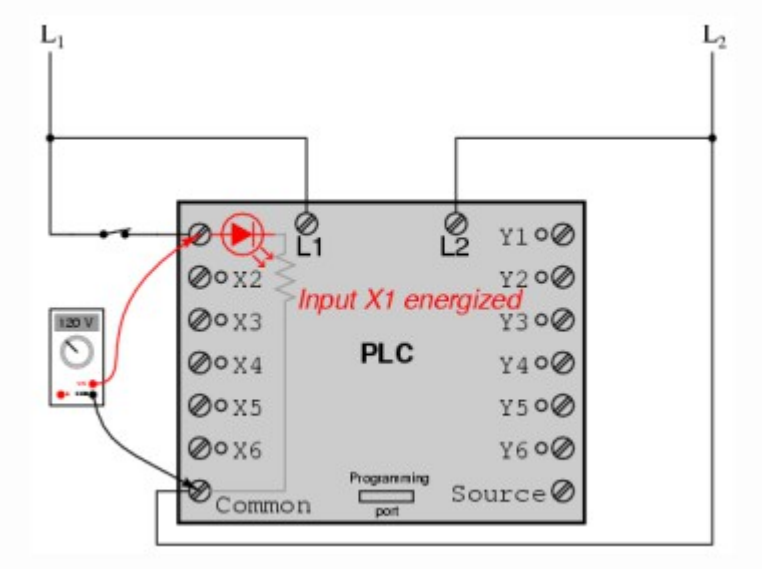

Рисунок 2 – Проверка напряжения входа ПЛК

Выходные сигналы генерируются компьютерной схемотехникой ПЛК, активируя переключающее устройство (транзистор, тиристор или даже электромеханическое реле) и связывая клемму «Источник» (правый нижний угол) с любым помеченным буквой Y выходом. Клемма «Источник» обычно связывается с L1. Так же, как и каждый вход, каждый выход, находящий под напряжением, отмечается с помощью светодиода, как показано на рисунке 3.

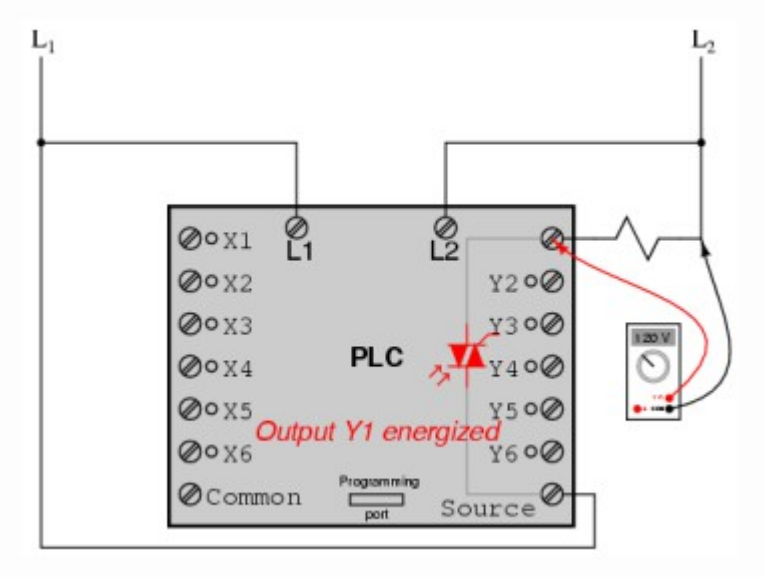

Рисунок 3 – Проверка напряжения выхода ПЛК

Таким образом, ПЛК может подключаться к любым устройствам, таким как переключатели и электромагниты.

Архитектура и память ПЛК

Обычно промышленный контроллер состоит из: центрального процессора, сетевых интерфейсов, модулей памяти и различных устройств ввода-вывода (Рисунок 4).

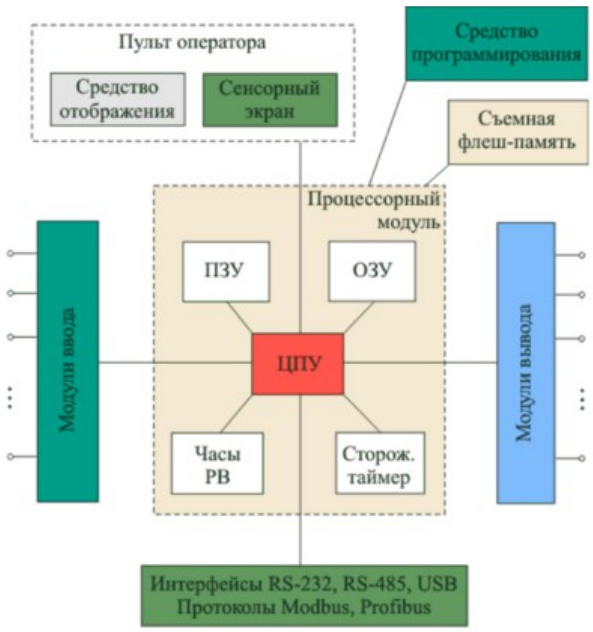

Рисунок 4 – Типовая схема устройства ПЛК

В состав процессорного модуля ПЛК входят следующие комплектующие: микропроцессор или ЦПУ (центральное процессорное устройство), часы реального времени, запоминающие устройства и watchdog.

К основным характеристикам микропроцессора относятся: тактовая частота, разрядность, поддержка портов для различных устройств ввода-вывода, архитектура, параметры работоспособности при определенных температурных диапазонах, выполнение операций с плавающей точкой, уровень потребляемой мощности.

Показатели производительности микропроцессоров с одинаковой архитектурой пропорциональны тактовой частоте. В большинстве контроллеров применяются микропроцессоры, реализованные на RISC (Reduced Instruction Set Computing) архитектуре, которые имеют сокращенное количество команд. В этом случае микропроцессор использует определенное число команд, обладающих одинаковой длиной, и множество регистров. Благодаря сокращенному набору команд можно создавать компиляторы с большими показателями эффективности, а также конвейер процессора, который за один такт может выдать результат выполнения действий одной из команд.

Промышленным контроллерам, имеющим дело с интенсивной математической обработкой данных, необходим математический сопроцессор (вспомогательный процессор, выполняющий операции с плавающей точкой) или же использование сигнальных процессоров, выполняющих математические операции на протяжении одного такта. Благодаря сигнальным процессорам достигается серьезное ускорение выполнения операций свертки или быстрого преобразования Фурье.

Емкость памяти характеризуется количеством переменных, которые можно обработать за время функционирования ПЛК. Время доступа к памяти микропроцессора – один из наиболее существенных показателей, способных ограничить быстродействие. Из-за этого происходит деление памяти на уровни иерархии, с учетом частоты и скорости использования, имеющихся в ней данных Иерархия памяти – это еще один существенный показатель архитектуры процессора, позволяющий снизить уровень возможного отрицательного воздействия медленной памяти на скорость работы микропроцессора.

Основные типы памяти промышленных контроллеров (ПЛК):

- ПЗУ – постоянное запоминающее устройство;
- ОЗУ – оперативное запоминающее устройство;
- Набор регистров.

Набор регистров - самые быстродействующие элементы памяти, так как их использует АЛУ (арифметико-логическое устройство) для выполнения простейших команд процессора. ПЗУ применяется как место хранения информации, которая редко подвергается изменению – операционная система, загрузчик, драйверы устройств, либо исполняемый модуль какой-либо программы. ОЗУ хранит в себе непосредственно данные подвергающиеся многократному изменению в период работы контроллера. К примеру, информация о проведении диагностики, отображаемые на дисплее переменные, значения тегов, промежуточные вычисления, выводимые на графики данные. В

роли ПЗУ (ROM - Read Only Memory), как правило, выступает перепрограммируемая электрически стираемая память (EEPROM - Electrically Erasable Programmable ROM). Кстати, флеш-память по сути - разновидность EEPROM. Принцип действия её заключается в хранении определенного заряда в конденсаторе, который образован подложкой МОП-транзистора и плавающим затвором. Главная особенность флеш памяти – её абсолютная энергонезависимость, т.е. возможность сохранения данных при отсутствии питания. Обновление данных во флеш-памяти происходит не отдельно взятыми ячейками, а за счет применения больших блоков. Все ПЗУ обладают большим недостатком - низким уровенем быстродействия

Количество циклов ввода информации во флеш-память ограничено всего несколькими десятками тысяч раз. Современными микропроцессорами в качестве ОЗУ применяется статическая память (SRAM - Static Random Access Memory), динамическая память (DRAM- Dynamic Random Access Memory), и синхронная динамическая память (SDRAM - Synchronous DRAM). Выполнение SRAM происходит на триггерах, которые способны хранить информацию неограниченно долгое время при условии наличия питания. Динамическая память промышленного контроллера сохраняет принадлежащие ей данные на конденсаторах, из-за чего требуется периодическая перезарядка конденсаторов. Основным недостатком триггерной памяти является высокий уровень стоимости и отношения цены к емкости. Это связано с тем, что на одном кристалле помещается относительно небольшое число триггеров. К достоинствам можно отнести высокий уровень быстродействия, исчисляемый гигагерцами, в то время как конденсаторная память не может преодолеть планку в несколько сотен герц. Все виды оперативной памяти отличаются тем, что при отсутствии питания все имеющаяся в них информация не сохраняется. Именно поэтому в некоторых типах ПЛК используется батарейное питание,

позволяющее сохранить работоспособность системы при условии кратковременного прерывания питания системы.

В модульных и моноблочных промышленных контроллерах используется параллельная шина, позволяющая обмениваться информацией с модулями ввода-вывода, благодаря чему быстродействие опроса значительно выше в сравнении с последовательной шиной. Виды параллельных шин: VME, PCI, ISA, CXM, ComactPCI, PC/104. Последовательная же шина, например, RS-485 необходима для подключения удаленных модулей ввода-вывода.

На рисунке 5 представлена общая аппаратная архитектура ПЛК

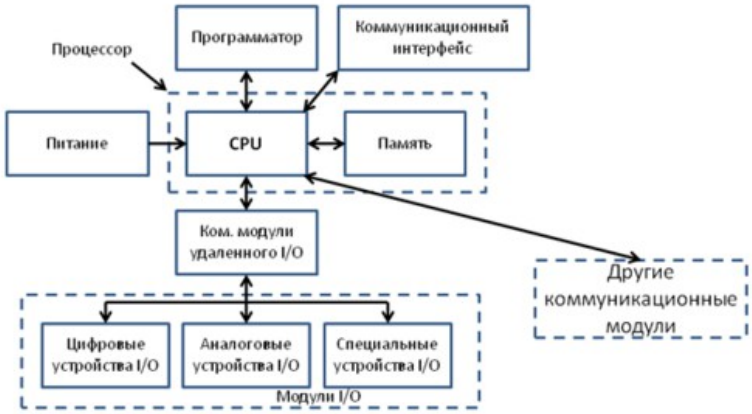

Рисунок 5 – Аппаратная архитектура ПЛК

Программная архитектура IEC 61131-3

Стандарт IEC 61131-3 задает программную модель, соответствующую современным принципам разработки программных продуктов. Эта модель характеризуется такими свойствами, как разработка «сверху-вниз», структурированное программирование, иерархическая организация, программные интерфейсы и инкапсуляция. К счастью, для того, чтобы эффективно программировать ПЛК, не нужно длительного обучения; хотя, конечно, будучи полностью реализованной, модель IEC 61131-3 весьма сложна.

Это основной недостаток модели, особенно по контрасту с простотой первых ПЛК.

Ниже описывается программная модель IEC 61131-3. Рассмотрим основные элементы подробнее (Рисунок 6).

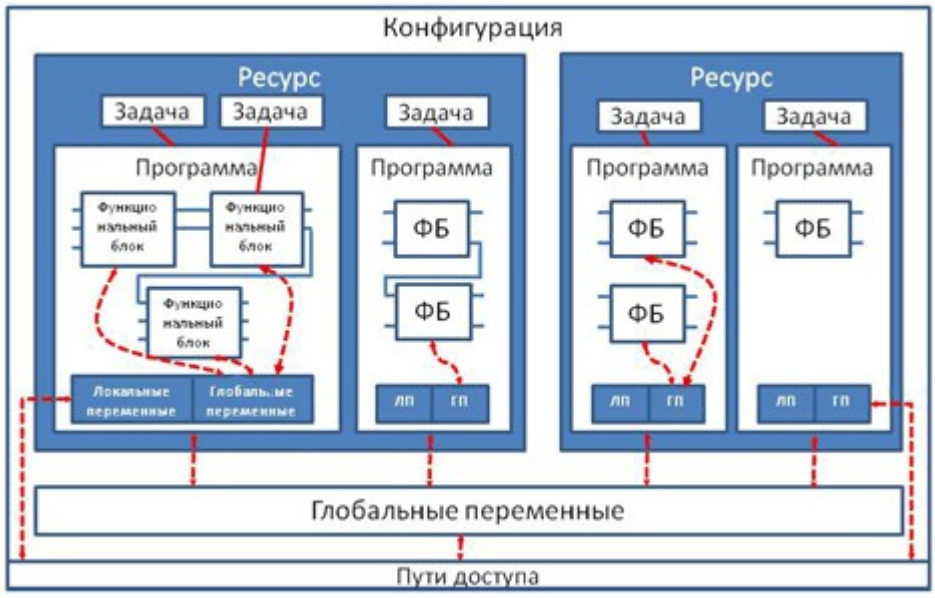

Рисунок 6 - Программная архитектура IEC 61131-3

Конфигурация (configuration) – совокупность всего ПО и данных, относящихся к системе ПЛК. В большинстве случаев, конфигурация относится к программам и данным одного ПЛК. В сложных системах, в которых взаимодействуют много различных ПЛК, у каждого есть своя конфигурация. Конфигурация обменивается данными с другими конфигурациями IEC в системе автоматизации через интерфейсы, называемые пути доступа (access paths). Использование слова «конфигурация» в этом контексте приводит к терминологическому конфликту, в связи с изначальным использованием этого термина в отрасли автоматизации как относящегося к таким объектам как модель процессора в ПЛК, коммуникационные интерфейсы, удаленный ввод/вывод, распределение памяти и т.д. Поэтому вендоры, производящие IEC-совместимые ПЛК, используют термин «конфигурация» в историческом

значении, и описывают совокупность программ и параметров термином проект (project).

Ресурс (resource) обеспечивает выполнение программ. Один или несколько ресурсов составляют конфигурацию. Обычно ресурс существует в границах ПЛК, но также он может существовать на ПК, допустим, для программного тестирования. Одна из основных функций ресурса – обеспечивать интерфейс между программой и физическим вводом/выводом ПЛК.

Программа (program) обычно состоит из взаимосвязанных функциональных блоков (function blocks), каждый из которых может быть написан на любом из языков IEC. Функциональные блоки или программы также называются программными модулями (program organization unit). Помимо функциональных блоков, в программе декларируются физические вводы/выводы и локальные переменные программы. Программа может считывать/вести запись в каналы ввода/вывода, глобальные переменные, и обмениваться данными с другими программами. Пути доступа позволяют конфигурациям обмениваться данными. Задачи (tasks) управляют выполнением программ или частей программ.

Задача управляет выполнением одной или нескольких программ и/или функциональных блоков. Выполнение программы подразумевает, что все функциональные блоки программы обрабатываются один раз. Выполнение функционального блока подразумевает, что все его программные элементы выполняются один раз. Для выполнения программы назначаются задачам, с настройкой для непрерывного, периодического выполнения или выполнения по триггеру (событию).

Переменные (Variables) задаются в различных программных элементах модели.

Локальная переменная (local variable) определяется программным элементом и доступна только ему. Локальные переменные могут быть заданы для функционального блока, программы, ресурса или конфигурации.

Глобальная переменная (global variable) заданная для конфигурации, ресурса или программы, доступна всем элементам, которые в них содержатся. Например, глобальная переменная конфигурации доступна всем программным элементам конфигурации. Глобальная переменная программы доступна всем функциональным блокам программы.

Обзор программного обеспечения для программирования ПЛК на языках МЭК 61131-3

Программирование ПЛК на языках МЭК 61131-3 осуществляется с помощью специализированного программного обеспечения, которое разрабатывается производителями ПЛК или фирмами, специализирующимися на создании ПО для систем автоматизации. Наиболее известными в мире являются системы CoDeSys фирмы 3S и ISaGRAF фирмы ICS Triplex.

Обзор инструментария CoDeSys

CodeSys – один самых развитых и полнофункциональных инструментов для программирования логических контроллеров на языках стандарта МЭК. Программируемые логические контроллеры (ПЛК) – устройства, автоматизирующие работу промышленных и бытовых приборов или производственных комплексов. ПЛК на физическом уровне являются устройствами, имеющими несколько дискретных и аналоговых входов и выходов, тогда как логика их работы закладывается программно в микрокомпьютерном ядре.

Название CodeSys является сокращением от «Controller Development System». Программный комплекс производится и распространяется немецкой фирмой 3S-Smart Software Solutions.

Несколько сот производителей устройств выпускают устройства автоматики с поддержкой программного интерфейса CoDeSys.

Для программирования доступны все стандартные МЭК языки: IL (Instruction List) - ассемблеро-подобный язык, ST (Structured Text) - Pascal-подобный язык, LD (Ladder Diagram) - язык релейных схем, FBD (Function Block Diagram) - язык функциональных блоков, SFC (Sequential Function Chart) - язык диаграмм состояний.

В качестве расширения стандарта МЭК в CoDeSys реализована поддержка объектно-ориентированного программирования, а также язык CFC (Continuous Function Chart), являющийся модифицированным.

Описание готового проекта хранится в одном единственном файле. Компилятор CoDeSys генерирует на основе проекта исполняемый машинный код, обеспечивая максимально возможное быстродействие прикладных программ. Поддерживаются различные процессоры известных марок. CoDeSys работает на всех 32х разрядных Windows. CoDeSys и контроллер взаимодействуют через промежуточное приложение Gateway-сервер. Он может работать как локально, так и удаленно через TCP/IP. Контроллеры подключаются к Gateway-серверу по протоколам RS232, TCP/IP или CAN.

Комплекс CoDeSys распространяется без лицензии и может быть установлен на нескольких рабочих местах.

Среда программирования CoDeSys

Среда программирования - это та часть, с которой непосредственно имеет дело пользователь (Рисунок 7). Она функционирует на ПК и является основным компонентом комплекса. Она включает редакторы для девяти языков программирования ПЛК, в том числе стандартные языки МЭК 61131-3. Пользователь может выбрать один из них и программировать простыми средствами либо задействовать всю мощь новейших инструментов CoDeSys. На

выходе CoDeSys непосредственно дает быстрый машинный код. Поддержаны все распространенные семейства микропроцессоров от 16 до 64-разрядных [1].

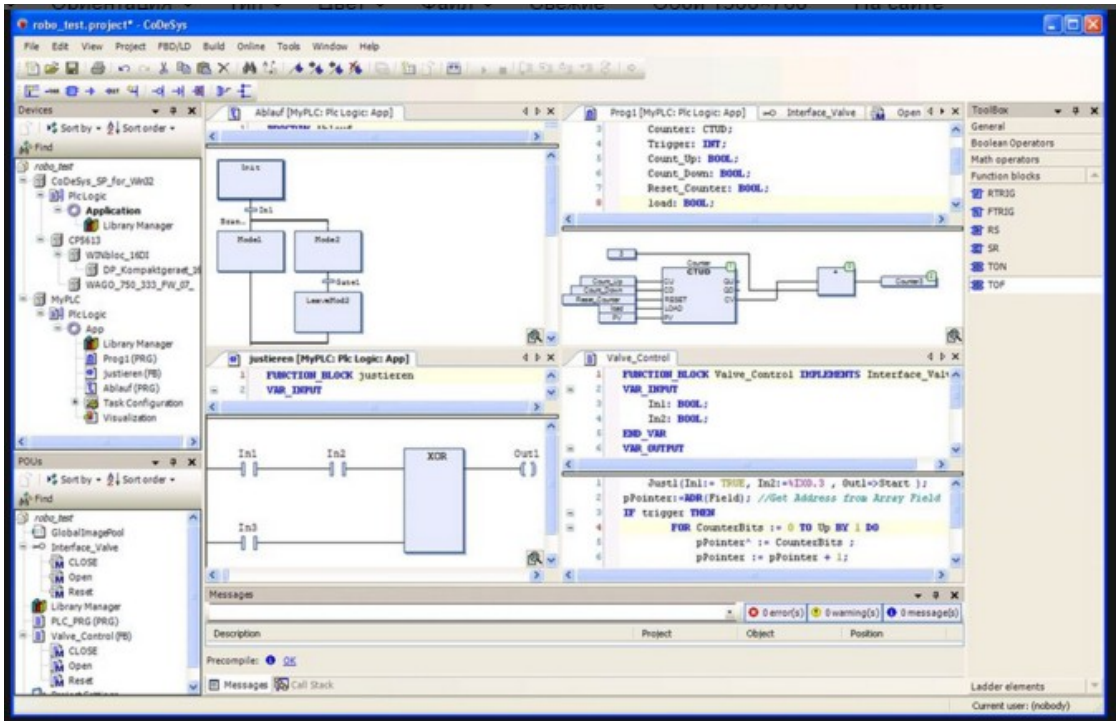

Рисунок 7 - Редактирование LD диаграммы в CoDeSys

Среда программирования CoDeSys включает набор инструментов для подготовки и отладки программ, компиляторы, конфигураторы, редакторы визуализации и т.д. При необходимости функциональность системы дополняется опциональными компонентами. Проект CoDeSys можно хранить не только на диске ПК, но и в контроллере, если он имеет достаточный объем памяти, что позволяет избежать потери исходных текстов или путаницы в проектах. Для больших проектов предусмотрено использование системы контроля версий (SVN), а сами схемы CoDeSys созраняте в файле формата .xml.

Для отладки пользователю не нужно открывать специальных отладочных окон или составлять каких-либо списков переменных. При подключении к ПЛК редакторы ввода программ "оживают". Непосредственно в них отображаются

значения всех видимых на экране переменных. Причем в сложных выражениях видны все промежуточные результаты [1].

Обзор технологии ISaGRAF

1.1.1.1 ISaGRAF — инструментальная графическая среда разработки прикладных программ для программируемых логических контроллеров (ПЛК) на языках стандарта IEC 61131-3 и IEC 61499, позволяющая создавать локальные или распределенные системы управления. Основа технологии — среда разработки приложений (ISaGRAF Workbench) и адаптируемая под различные аппаратно-программные платформы исполнительная система (ISaGRAF Runtime). В настоящее время ISaGRAF производится и распространяется компанией Rockwell Automation.

ISaGRAF позволяет ускорить разработку и внедрение приложений, уменьшить время выхода на рынок и предоставляет конкурентоспособную аппаратно-программную независимость.

ISaGRAF сохраняет схемы в формате .xml.

ISaGRAF - это встраиваемая, масштабируемая технология программирования контроллеров, позволяющая создавать как приложения для автономных контроллеров, так и распределенные приложения для нескольких обменивающихся данными по сети контроллеров. ISaGRAF состоит из трех взаимосвязанных компонент (Рисунок):

- система разработки приложений (Workbench) - для проектирования, компиляции, симуляции, загрузки приложения в контроллер и отладки
- встраиваемая целевая система - легко переносимый машинно-независимый программный комплекс, который встраивается в контроллер и исполняет приложения, спроектированные в системе разработки приложений
- средства разработчика - для написания драйверов под ISaGRAF, переноса целевых систем на другие аппаратные и программные платформы и т.д.

-

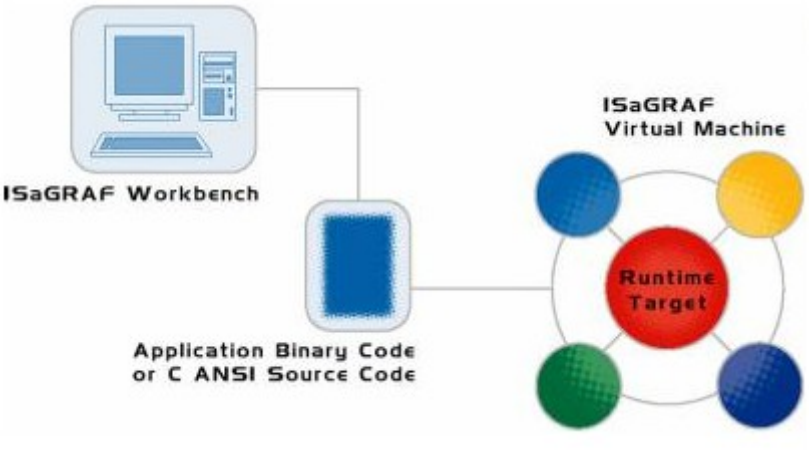

Рисунок 8 – Схема работы ISaGRAF

Обзор возможностей ISaGRAF 5

В версии ISaGRAF 5.0 реализована поддержка нового типа функциональных блоков, определяемых стандартом IEC 61499. В ISaGRAF реализован ряд расширений спецификации стандарта IEC 61131-3, в частности язык Flow Chart. Инструментальная система разработки приложений обеспечивает проект всеми возможностями языков стандарта IEC61161. Устойчивость к сбоям, способность обрабатывать большое количество точек ввода/вывода позволяют с успехом применять ISaGRAF как в небольших встраиваемых приложениях, так и в распределенных проектах автоматизации. В многозадачной системе могут исполняться несколько исполнительных систем ISaGRAF PRO с различными циклами опроса. Механизм связывания переменных обеспечивает передачу данных между исполнительными системами как в локальном, так и в распределенном проекте, причем аппаратные платформы могут работать под управлением различных операционных систем.

Обзор языка программирования релейно-лестничной логики стандарта IEC 61131-3

Язык релейных диаграмм LD (Ladder Diagram) или релейно-контактных схем (РКС) – графический язык, реализующий структуры электрических цепей. РКС - это американское изобретение. В начале 70-ых гг. XX в. релейные автоматы сборочных конвейеров начали постепенно вытесняться программируемыми контроллерами. Некоторое время те и другие работали одновременно и обслуживались одними и теми же людьми. Так появилась задача прозрачного переноса релейных схем в ПЛК. Различные варианты программной реализации релейных схем создавались практически всеми ведущими производителями ПЛК. Благодаря простоте представления РКС обрел заслуженную популярность, что и стало основной причиной включения его в стандарт МЭК.

Слово «релейная логика» звучит сегодня достаточно архаично, почти как «ламповый компьютер». Тем более в связи с созданием многочисленных быстродействующих и надежных бесконтактных (в частности оптоволоконных) реле и мощных переключающих приборов, таких как мощные полевые транзисторы, управляемые тиристоры и приборы IGBT. Но несмотря на это, релейная техника все еще очень широко применяется [4].

Графически LD-диаграмма представлена в виде двух вертикальных шин питания. Между ними расположены цепи, образованные соединением контактов (Рисунок 9). Нагрузкой каждой цепи служит реле. Каждое реле имеет контакты, которые можно использовать в других цепях.

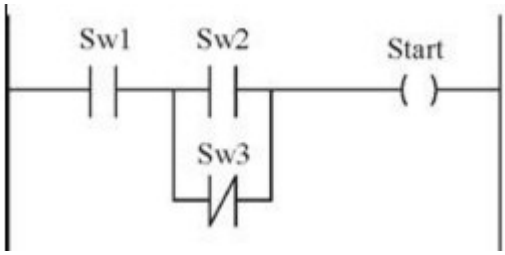

Рисунок 9– Схема LD из одной цепи.

Логическое последовательное (И), параллельное (ИЛИ) соединение контактов и инверсия (НЕ) образуют базис Буля. В результате LD идеально подходит не только для построения релейных автоматов, но я для программной реализации комбинационных логических схем. Благодаря возможности включения в LD функций и функциональных блоков, выполненных на других языках, сфера применения языка практически не ограничена [5].

Техническое задание на разработку программной системы

Общие сведения

Назначение программного обеспечения – это упрощение работы с ПЛК. Целью создания данной системы является мгновенная реакция на изменения в программах на языке LD, которая не требует повторной компиляции программного кода и транслирует его «на лету», а редактирование программного кода возможно на самом ПЛК [6].

Функциональное назначение

Система предназначена для интерпретации программ на языке релейно-лестничной логики (далее LD).

Описание логической структуры

После запуска графического редактора инженер составляет программу на языке LD. Затем инженер сохраняет программу в формате xml. Дальше инженер запускает интерпретатор который выполняет программу и выдает результат.

Общие сведения

Полное наименование системы и ее условное обозначение

Полное наименование системы: Программная система для редактирования и валидации программ на языке релейно-лестничной логики.

Плановые сроки начала и окончания работы по созданию системы

Плановый срок начала работы по разработки системы: 30 марта 2016 года. Плановый срок окончания работ: 17 июня 2016 года.

Назначение системы

Назначение системы

Система предназначена для интерпретации программ на языке релейно-лестничной логики (далее LD).

Основным назначением системы является ускоренная трансляция изменений в программном коде на языке LD.

Требования к системе

Требования к структуре и функционированию системы

В системе выделены следующие функциональные подсистемы:

- графический векторный редактор для создания и редактирования логической цепи с замкнутыми и разомкнутыми контактами;
- транслятор XML;
- интерпретатор исходного кода.

Требования к эргономике и технической эстетике

Графический редактор должен обеспечивать удобный для конечного пользователя интерфейс, отвечающий следующим требованиям.

- размеры графики должны соответствовать разрешению экрана;
- цвет шрифта должен быть черным;
- «холст» редактора должен быть белого цвета и иметь привязку к сетке;
- при возникновении ошибки, должен выводиться диалог с ошибкой и рекомендациями по ее устранению;
- информационные сообщения должны выводиться как всплывающие сообщения.

Требования к функциям, выполняемым системой

Графический векторный редактор

- добавление, редактирование и удаление контактов на панели редактирования исходного кода («холсте»);
- добавление связей между контактами;
- изменение свойств контактов;
- сохранение файла с исходным кодом в формате XML.

Интерпретатор LD

- разбор файла на языке XML;
- трансляция исходного кода в машинный код;
- выполнение программы;
- вывод результатов.

Требования к видам обеспечения

Требования к математическому обеспечению

Не предъявляются.

Требования к лингвистическому программному обеспечению

При реализации системы должны использоваться следующие языки высокого уровня: Java и Python.

Для организации диалога системы с пользователем должен применяться графический интерфейс.

Требования к техническому обеспечению

Приложение должно использоваться на программируемых логических контроллерах под управлением ОС Linux.

Средства разработки программной системы

IntelliJ IDEA 15.0.3

Краткая характеристика средства

Интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Руководство по настройке средства

Для работы среды необходимо установить JDK той версии, с которой будет производиться работа.

Требования по настройке средства

Среда работает под операционными системами Windows и Linux.

PyCharm 4.5.3

Краткая характеристика средства

Интегрированная среда разработки для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент

для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией JetBrains на основе IntelliJ IDEA..

Руководство по настройке средства

Для работы среды необходимо установить Python той версии, с которой будет производиться работа.

Требования по настройке средства

Среда работает под операционными системами Windows, Mac OS X и Linux.

Принципы построения интерпретатора

Интерпретатор — это программа, которая воспринимает входную программу на исходном языке и выполняет ее. В свою очередь текст исходной программы это, промежуточный код, являющийся результатом работы транслятора XML. Выходом интерпретатора является результат выполнения инструкций исходной программы.

Процесс трансляции разделен на два этапа:

- Транслятор XML;
- Интерпретация промежуточного кода.

Данная структура изображена на рисунке 10.

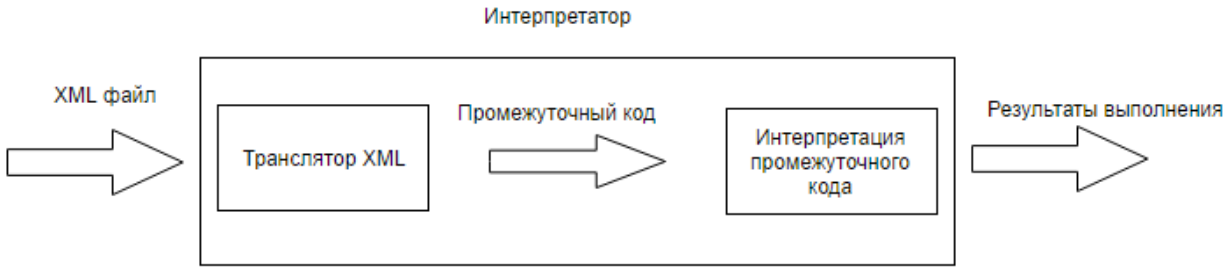

Рисунок 10 – Структура интерпретатора

Интерпретация промежуточного кода представляет собой транслятор инструкций на некотором промежуточном языке и разделена на три основных фазы:

- Лексический анализ;
- Синтаксический анализ;
- Выполнение программы.

Фазы трансляции данного интерпретатора изображены на рисунке 11.

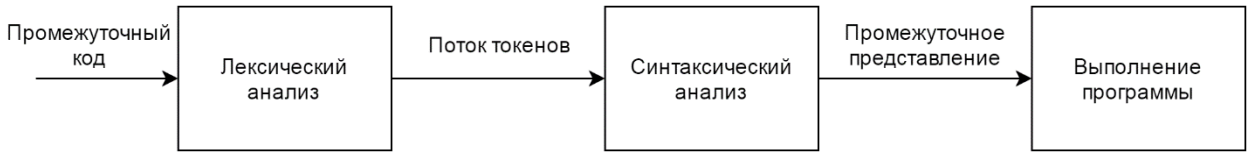

Рисунок 11 – Фазы трансляции

Транслятор XML

Транслятор является одним из этапов средства интерпретации исходного кода. Этот этап необходим для преобразования входящего файла, который имеет структуру XML. Для разработки транслятора была выбрана библиотека

В данной работе в качестве транслированной формы выступает промежуточный код, который имеет форму польскую нотации.

Польская нотация

Польская нотация – это форма записи логических, арифметических и алгебраических выражений. Характерная черта такой записи — оператор располагается слева от операндов. Если оператор имеет фиксированную арность, то в такой записи будут отсутствовать круглые скобки, и она может быть интерпретирована без неоднозначности. По той же причине что при

программировании на ПЛК с помощью языка LD используется арность выражений удобно использовать именно эту нотацию.

Использование польской нотации по отношению к LD, позволяет представить контакты и обмотки в виде такого выражения, в котором слева записано наименование функции (например, логическое умножение), а далее идет список ее аргументов. Пример такого преобразования представлен на рисунке 12.

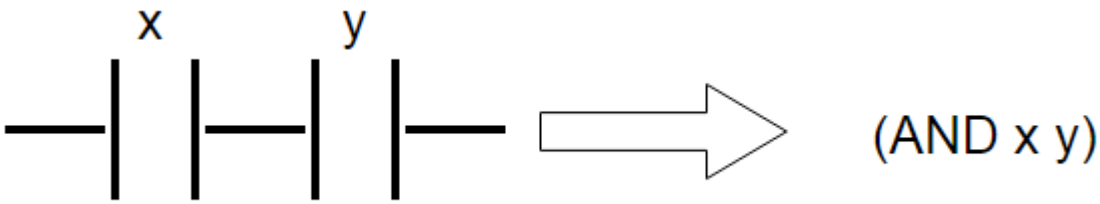

Рисунок 12 – Пример преобразования контакта в промежуточный код

Дальнейшая работа заданной функции определяется правилами ее выполнения в ходе интерпретации программы.

Лексический анализ

Лексический анализатор (известен также как сканер) осуществляет чтение входной цепочки символов и их группировку в элементарные конструкции, называемые лексемами (рисунок 13) [7].

Рисунок 13 – Группировка входной цепочки в лексемы

Каждая лексема имеет класс и значение. Обычно претендентами на роль лексем выступают элементарные конструкции языка, например, идентификатор, действительное число, комментарий. Полученные лексемы передаются синтаксическому анализатору. Сканер не является обязательной частью транслятора. Однако он позволяет повысить эффективность процесса трансляции.

Лексический анализатор представляет собой первую фазу транслятора, а его основная задача состоит в чтении входных символов исходной программы, их группировании в лексемы и вывод последовательностей токенов для всех лексем исходной программы. Поток токенов пересылается синтаксическому анализатору для разбора (рисунок 14).

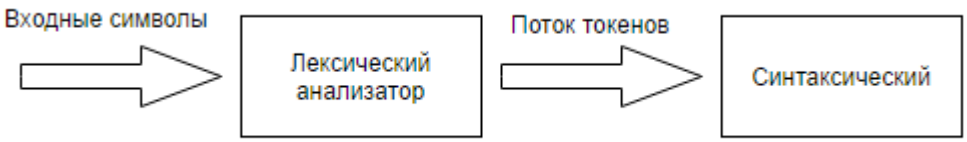

Рисунок 14 – Схема лексического анализа

Компонент транслятора, отвечающий за лексический анализ, называется лексическим анализатором (сканером, токенайзером, лексером).

Поскольку лексический анализатор является частью транслятора, которая читает исходный текст, он может заодно выполнять и некоторые другие действия, помимо идентификации лексем. Одной из таких задач является отбрасывание комментариев и пробельных символов (пробел, символы табуляции и новой строки, а также, возможно, некоторые другие символы, использующиеся для отделения токенов друг от друга во входном потоке). Еще одной задачей является синхронизация сообщений об ошибках, генерируемых

компилятором, с исходной программой. К примеру, во многих компиляторах лексический анализатор отслеживает количество символов новой строки, чтобы каждое сообщение об ошибке сопровождалось номером строки, в которой она обнаружена. Также в некоторых компиляторах лексер создает копию исходной программы с сообщениями об ошибках, вставленными в соответствующие места исходного текста. Если исходная программа использует макропроцессор, то раскрытие макросов также может выполняться лексическим анализатором.

При рассмотрении лексическою анализа используются три связанных, но различных терминами:

- Токен (token) представляет собой пару, состоящую из имени токена и необязательного списка атрибутов. Имя токена - это абстрактный символ, представляющий тип лексической единицы, например конкретное ключевое слово, последовательность входных символов, составляющую идентификатор или знак операции. Имена токенов являются входными символами, обрабатываемыми синтаксическим анализатором. Далее обычно мы будем записывать имя токена полужирным шрифтом и ссылаться на токен по его имени.

- Шаблон (pattern) - это описание вида, который может принимать лексема токена. В случае ключевого слова шаблон представляет собой просто последовательность символов, образующих это ключевое слово. Для идентификаторов и некоторых других токенов шаблон представляет собой более сложную структуру, которой соответствуют (matched) наборы строк.

- Лексема (lexeme) представляет собой последовательность символов исходной программы, которая соответствует шаблону токена и идентифицируется лексическим анализатором как экземпляр токена.

Во многих языках программирования описанные далее ситуации охватывают большинство (если не все) токенов.

1. По одному токену для каждого ключевого слова. Шаблон для ключевого слова выглядит так же, как само ключевое слово.

2. Токены для операторов, либо отдельные, либо объединенные в классы

3. Один токен, представляющий идентификаторы.

4. Один или несколько токенов, представляющих константы, такие как числа и строковые литералы.

5. Токены для каждого символа пунктуации, такие как левые и правые скобки, запятые или точки с запятыми.

Если шаблону могут соответствовать несколько лексем, лексер должен обеспечить дополнительную информацию о лексемах для последующих фаз трансляции.

Синтаксический анализ

В интерпретаторе синтаксический анализатор получает строку токенов от лексического анализатора и проверяет, может ли эта строка порождаться грамматикой исходного языка. Он также сообщает о всех выявленных ошибках. причем достаточно внятно и полно. Кроме того, он должен уметь обрабатывать обычные часто встречающиеся ошибки и продолжать работу с оставшейся частью программы [7].

Имеется три основных типа синтаксических анализаторов грамматик. Универсальные методы разбора, обычно применяемые в компиляторах, классифицируются как:

- нисходящие (сверху вниз, top-down)

- восходящие (снизу вверх, bottom-up)

- универсальные.

Как явствует из названий, нисходящие синтаксические анализаторы строят дерево разбора сверху (от корня) вниз (к листьям), тогда как восходящие начинают с листьев и идут к корню. В обоих случаях входной поток синтаксического анализатора сканируется посимвольно слева направо.

Наиболее эффективные нисходящие и восходящие методы работают только с подклассами грамматик, однако некоторые из этих подклассов, такие как LL- и LR-грамматики, достаточно выразительны для описания большинства синтаксических конструкций языков программирования. Реализованные вручную синтаксические анализаторы чаще работают с LL-грамматиками. Синтаксические анализаторы для несколько большего класса LR-грамматик обычно создаются с помощью автоматизированных инструментов.

Разработка программной системы

Перед началом разработки программной системы, была спроектирована ее структура (рисунок 15).

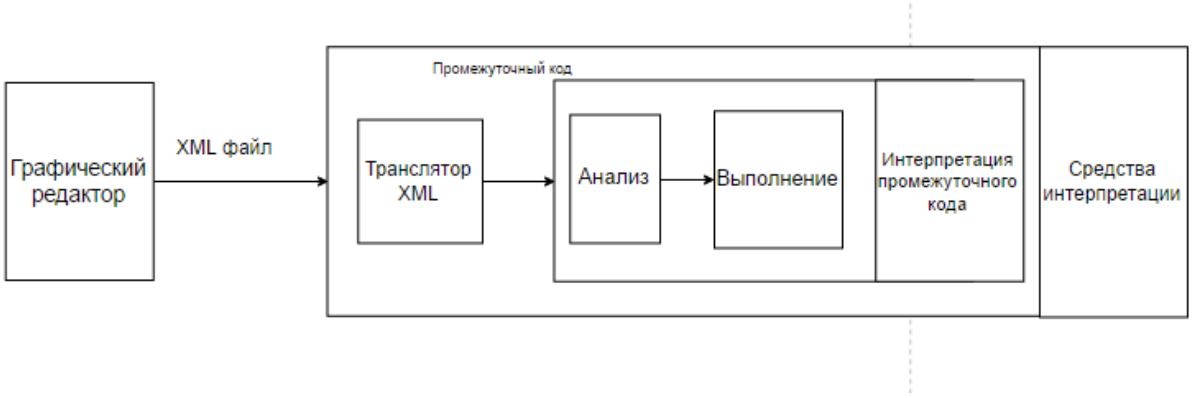

Рисунок 15 – Структура программной системы

Так как язык, для которого разрабатывался интерпретатор, является прежде всего графическим, в первую очередь необходим графический редактор,

который позволит создавать и редактировать исходный код на языке LD. В качестве формата файла для представления исходного кода на языке LD был выбран XML, в связи с тем, ведущие системы, которые работают с графическими языками сохраняют схемы формате XML. Расширение файла - .ld.

После того, как создан файл, содержащий в себе исходный код в виде структуры XML, следующим этапом является сам процесс интерпретации исходного кода на языке LD, который разделяется на несколько основных стадий. Эти стадии позволяют разобрать XML-файл с исходным кодом, преобразовать его к промежуточному коду, провести ряд фаз по анализу и выполнить его.

Графический редактор

Данный модуль разработан на языке Java с применением библиотеки JGraphX.

Графический векторный редактор позволяет инженеру АСУ ТП создавать программы с помощью графического языка программирования LD. Редактор имеет понятный пользовательский интерфейс и во многом похож на популярные средства разработки программ на языке LD (рисунок 16).

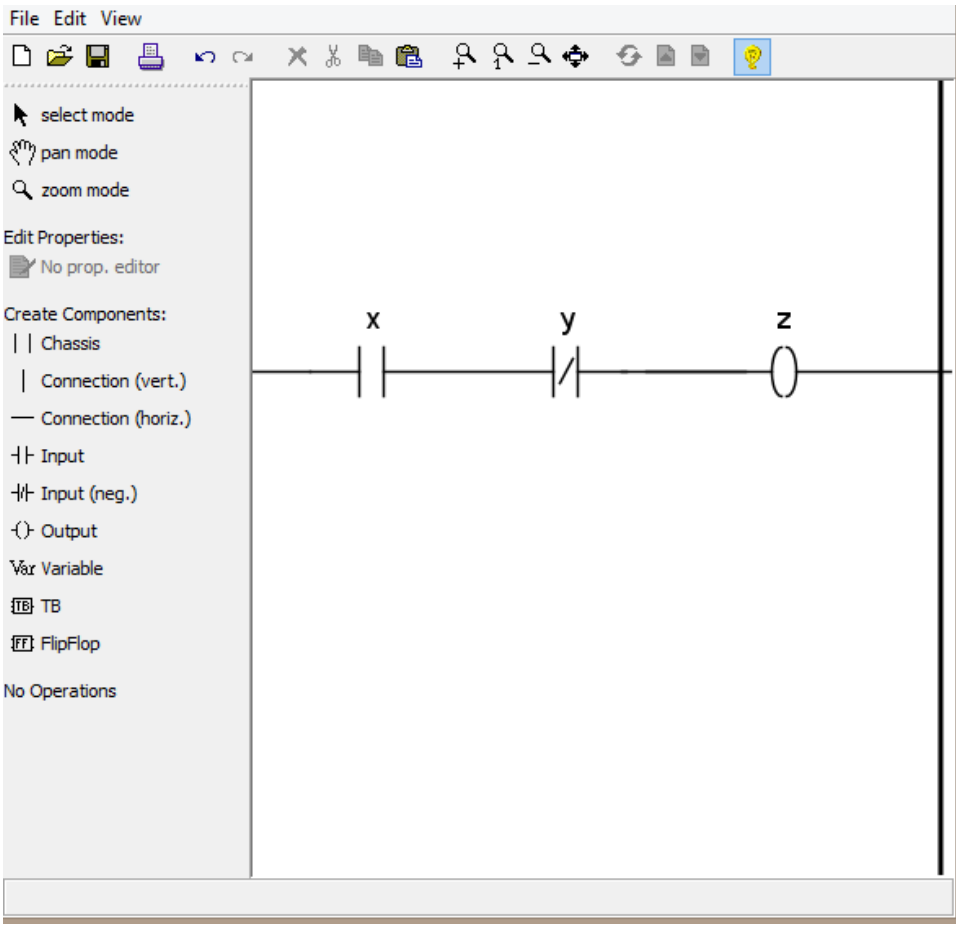

Рисунок 16 – Пользовательский интерфейс графического редактора

Графический редактор имеет три основных функциональных компонента:

- Блок вариантов работы с холстом - содержит список курсоров, которыми можно работать с холстом.
- Область редактирования (холст) – элемент, на котором можно создавать и редактировать программный код на языке LD путем перемещения блоков и создания связей между ними.
- Блок компонентов – содержит перечень всех реализованных контактов и обмоток.

После создания сценария на языке LD его можно сохранить в формате .ld который понятен интерпретатору и имеет структуру XML.

Структура XML-файлов

В XML-файле существуют три основных тега:

1) invar – содержит в себе информацию о конкретной переменной, которая подается на вход. Включает в себя два дополнительных тега: val и name. val на значение, а name хранит в себе имя этой переменной. Если в коде на языке LD используется несколько входных переменных, то будет создано и несколько таких конструкций с данным тегом.

2) outvar – хранит информацию обо всех переменных, в которых хранится результат. Так же, как и invar содержит внутри себя такие теги, как val и name. Если в коде на языке LD используется несколько переменных выхода, то будет создано и несколько таких конструкций с данным тегом

3) contact – тег, который содержит информацию об одном контакте или обмотке. Содержит в себе следующие теги:

- index – уникальный идентификатор контакта или обмотки, который используется для взаимодействия с другими контакта или обмотками;

- contype – содержит в информацию о том, какая функция должна применяться к переменным данного контакта или обмотки;

- inputvar – указывает что подается на вход функционального блока. Может быть использован несколько раз, если на вход подается несколько переменных. Если на вход подается результат другого блока, то используется индекс этого блока, а также знак «-» перед этим индексом (например, -3);

- outputvar – указывает на переменные или блоки, которым передается результат применения функции к переменным данного блока.

Пример представления программы на языке функциональных блоковых диаграмм в виде структуры XML приведен на рисунке 17.

```
<board>
    <outvar>z</outvar>

    <invar>
        <name>i</name>
        <val>nil</val>
    </invar>


    <contact>
        <index>1</index>
        <contype>coni</contype>
        <invar>x</invar>
        <outvar>2</outvar>
    </contact>

    <contact>
        <index>2</index>
        <contype>coni</contype>
        <invar>a</invar>
        <invar>-1</invar>
        <outvar>z</outvar>
    </contact>

</board>
```

Рисунок 17 – Представление релейной схемы в формате XML

Транслятор XML

После того, как файл с исходным кодом создан, его можно передать средству интерпретации для анализа и выполнения.

Транслятор XML – это одна из первых стадий трансляции, которая позволяет разобрать XML-файл и получить сущности, необходимые для построения промежуточного кода, который будет понятен интерпретатору.

Транслятор разработан на языке Python версии 2.7. Для разбора XML использована библиотека lxml версии 3.6.0, так как в ней уже заложены принципы разбора XML файлов, что в свою очередь упрощает работу с этим типом файлов.

В качестве промежуточного кода используется польская нотация. Все преобразования XML-файла к промежуточному коду ведутся по нескольким правилам.

Чтобы построить инструкцию на промежуточном коде, которая в дальнейшем будет запрашивать у пользователя значение входной переменной, транслятор сначала находит все теги invar, а затем выстраивает следующую конструкцию для каждого такого тега: (input contype var), где input – функция, которая запрашивает у пользователя значение переменной с клавиатуры, а затем добавляет эту переменную в таблицу символов с указанным значением, contype var – тип и имя переменной, которые будут привязаны к введенному пользователем значению и добавлены в таблицу символов.

Для построения инструкции, которая вычисляет значение переменной выхода, необходимо найти все имена переменных, которые заключены между тегом outvar, а затем для каждой переменной найти функциональный блок, который записывает результат вычисления своей функции в эту переменную. Запись будет выглядеть подобным образом: (output type var *), где output – функция, которая присваивает переменной type var значение, которое будет вычислено всеми функциями, которые будут записаны вместо символа «*».

Руководство пользователя

Графический редактор

Графический редактор является важной частью программной системы и по этому рассмотрим сперва его.

Для составления релейной схемы нужны выделять те контакты и обмотки, которые нужны для работы. Для этого нужно выбрать в блоке компонентов нужный элемент и перетянуть его на холст и соединить их в цепь (Рисунок 18).

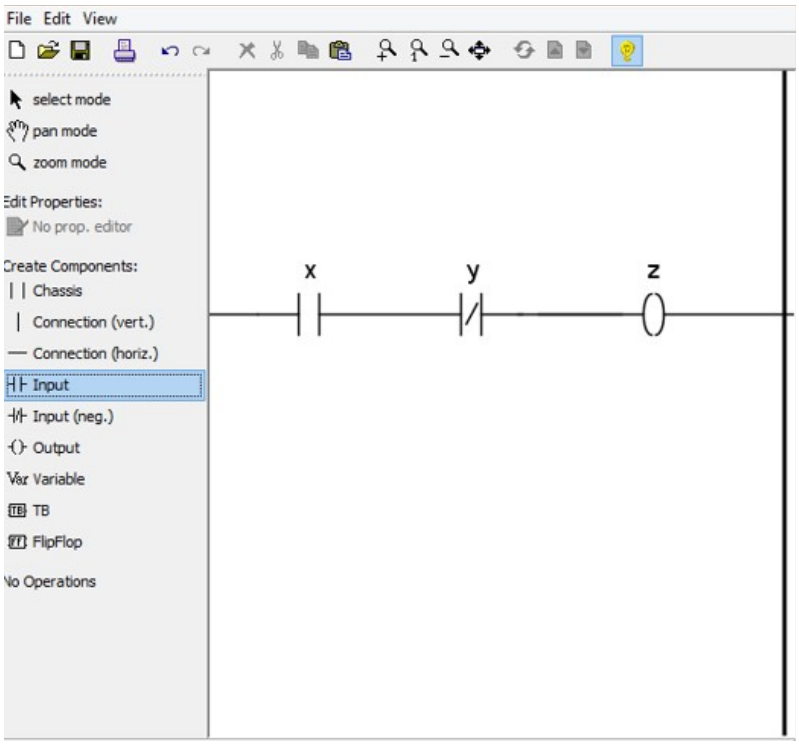

Рисунок 18 – Составление схемы на языке LD

Компонент сразу же появится на холсте.

Для переименования контактов и обмоток нужно выбрать тот контакт или отмотку, которая нужна (Рисунок 19) и нажать кнопку «Edit Propeties» и ввести новое название (Рисунок 20).

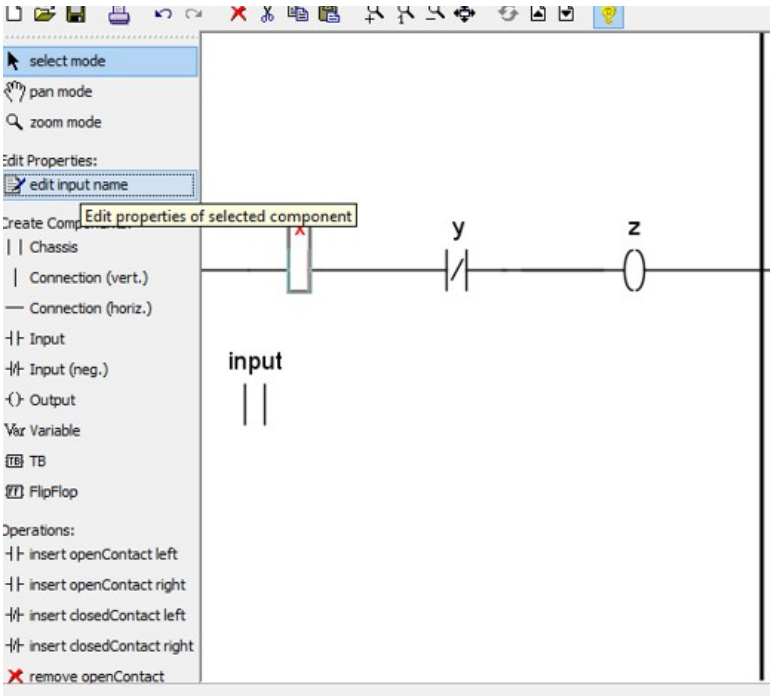

Рисунок 19– Переименовывание контакта

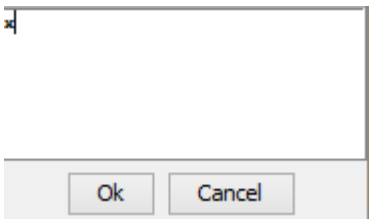

Рисунок 20 – Окно введения нового имени

Для сохранения схемы нужно нажать кнопу сохранить, и появится окно сохранения и введения имени.

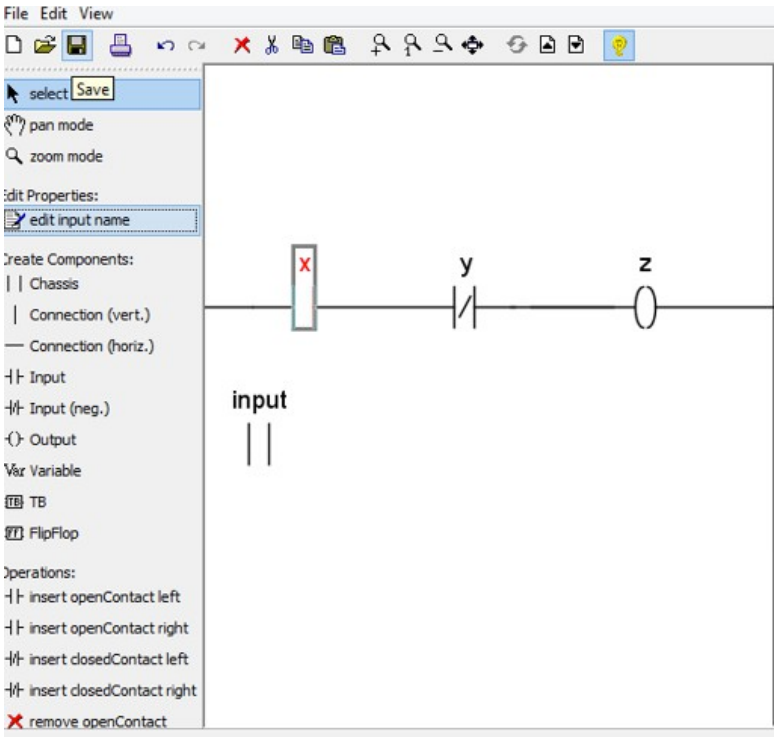

Рисунок 21 – Сохранение файла

ЗАКЛЮЧЕНИЕ

В ходе данной работы был произведен обзор существующих средств трансляции языков стандарта IEC 61131-3. Было выяснено, что программные инструменты, которые являются интерпретаторами данных языков, практически отсутствуют.

Была описана структура программной системы для редактирования и валидации программ на языке релейно-лестничной логики (LD)

В ходе данной работы был разработан графический редактор, позволяющий редактировать исходный код на графическом языке LD.

Так же был разработан интерпретатор программ на языке LD. Который включал в себя транслятор XML, фазы лексического и синтаксического анализа, а также процесс выполнения программы.

Приложение разработана в средах разработки IntelliJ IDEA и PyCharm на языках программирования Java и Python.

2 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Петров, И.В. Инструментарий для программирования контроллеров / И.В. Петров Автоматизация в промышленности. – 2012. – №8.
2. IEC 61131-3, 2 ed. Programmable Controllers – Programming Languages. – Введ. 2001 International Electrotechnical Commission, 2001. – 156 с.
3. Принципы работы и основы программирования ПЛК [Электронный ресурс]: – Режим доступа: http://electrik.info/main/automation/710-princip-raboty-i-osnovyprogramirovaniya-plk.html
4. Рахимбердиев, А Современные процессы разработки программного обеспечения [Электронный ресурс]: – Режим доступа: http://www.rsdn.ru/ article/ Methodologies/SoftwareDevelopmentProcesses.xml. – (дата обращения: 18.05.2016).
5. Хаф, Л. Методологии разработки ПО [Электронный ресурс] : – Режим доступа: http://www.lib.csu.ru/dl/bases/prg/kompress/articles/2003_10_XP/. – (дата обращения: 18.05.2016).
6. Современные процессы разработи программного обеспечения [Электронный ресурс]: – Режим доступа: http://rsdn.ru/article/Methodologies/SoftwareDevelopmentProcesses.xml (дата обращения: 18.05.2016).
7. Костельцев, А.В. Построение интерпретаторов и компиляторов / А.В. Костельцев. – СПб.: Наука и Техника, 2001. – 224 с.
8. Кузнецов, А.С. Конспект лекций по Основам построения трансляторов. [Электронный ресурс]. - Режим доступа: https://e.sfu-kras.ru/my/.

9. Ахо, А.В. Компиляторы. Принципы, технологии и инструментарий: учебное пособие / А.В. Ахо, М.С. Лам, Р. Сети. – Вильямс, 2008. – 1184 с.